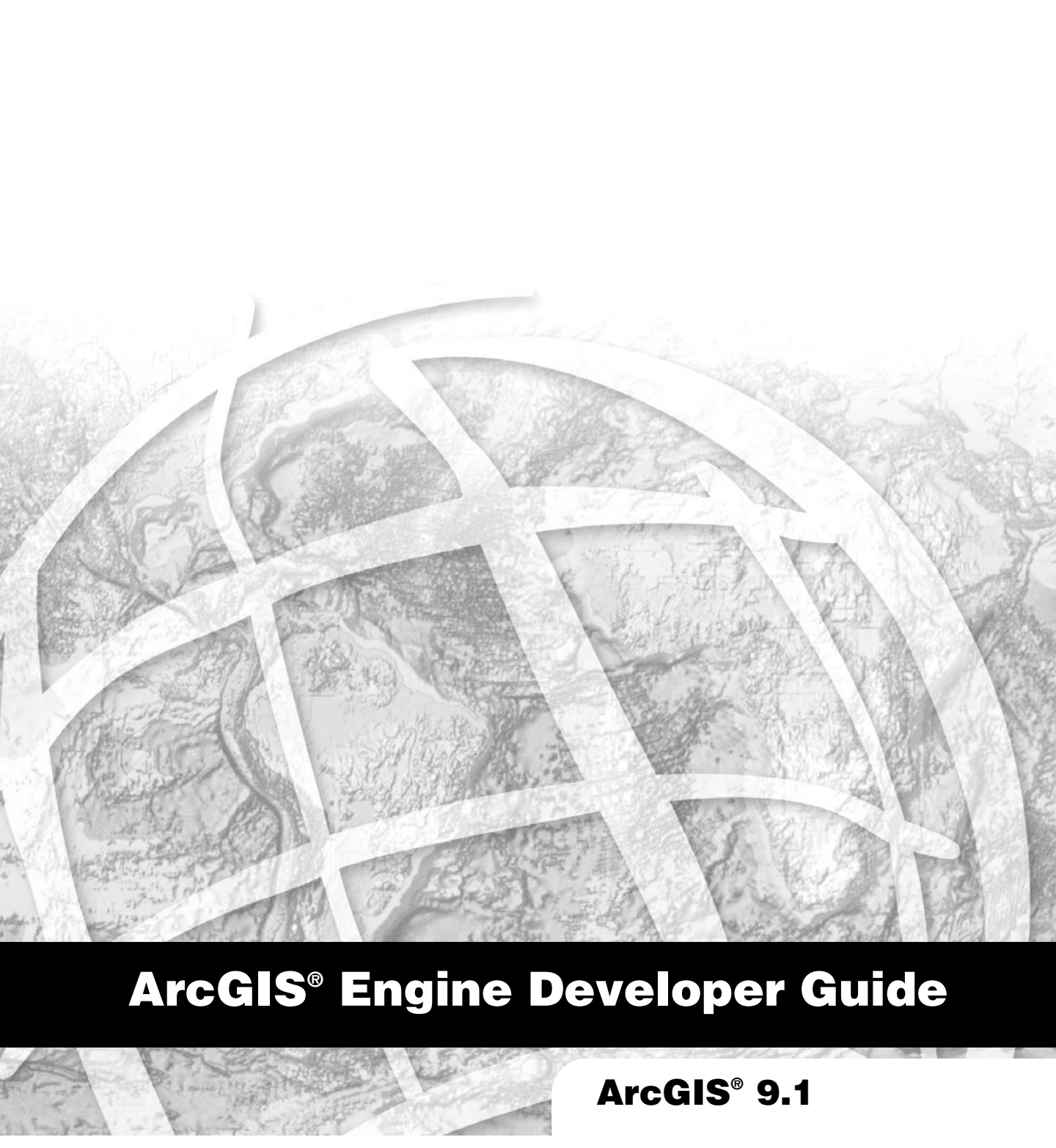


ArcGIS® Engine Developer Guide

ArcGIS® 9.1

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

Contributing Writers

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

Contents

1 Introducing ArcGIS Engine

ESRI® ArcGIS® Engine is a platform for building custom standalone geographic information system (GIS) applications that 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

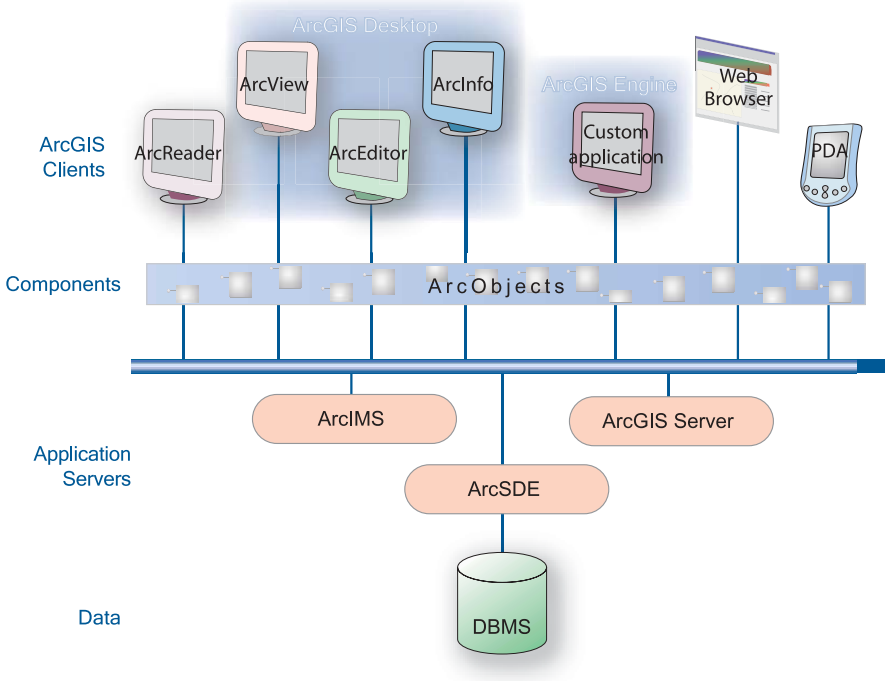
Developers wanting to customize ArcGIS Desktop applications or work with ArcGIS Server should refer to the ArcGIS Desktop Developer Guide *and the* ArcGIS Server Administrator and Developer Guide.

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 want 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.



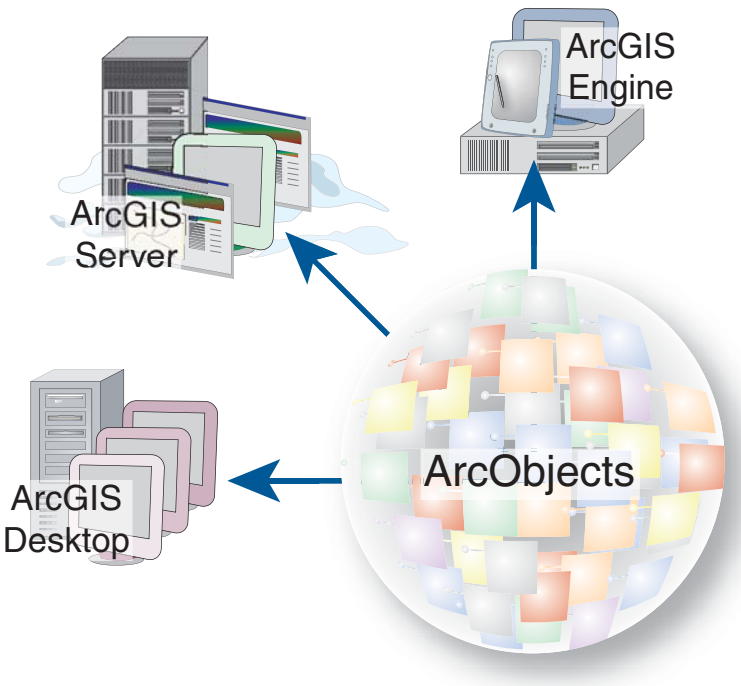
Each of the GIS frameworks also includes the ArcSDE® gateway, an interface for managing geodatabases in numerous relational database management systems (RDBMS).

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. ArcGIS Engine and ArcGIS Server, developer-centric products, make 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.

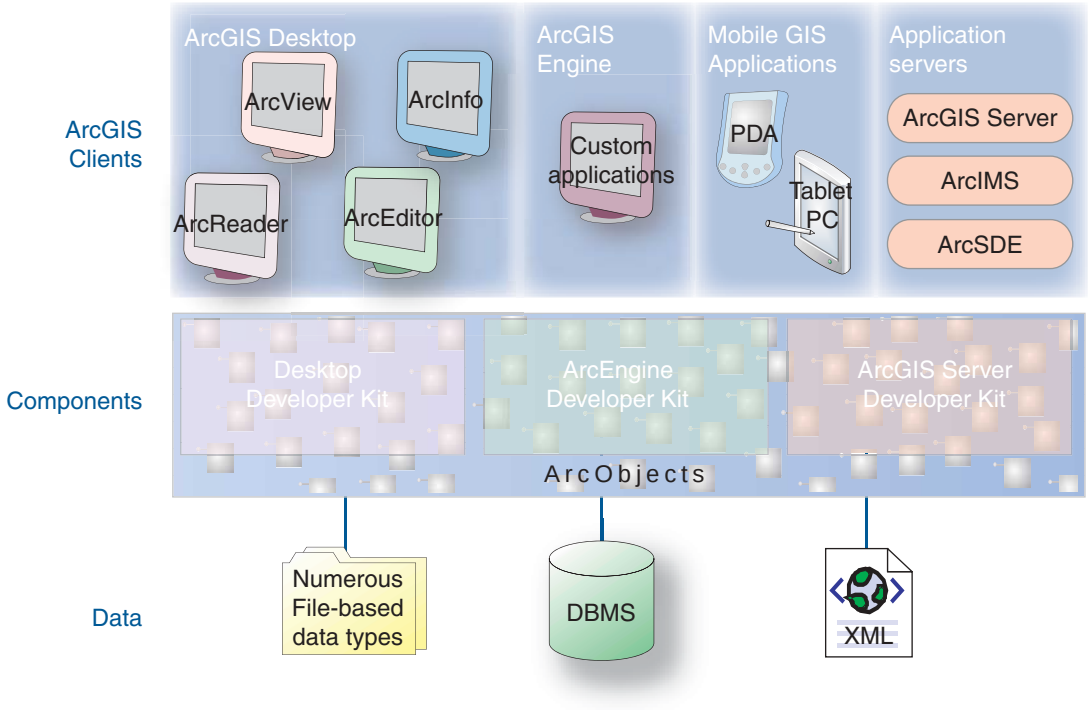


ArcGIS 9 has a common developer experience across all ArcGIS products (Engine, Server, and Desktop). You, as a developer, can work with ArcObjects using standard programming frameworks to extend ArcGIS Desktop, build custom applications with ArcGIS Engine, and implement enterprise GIS applications using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

As noted previously, this book focuses on building and deploying custom applications using ArcGIS Engine. If you want 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++, Component Object Model (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 (UI) 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 ArcGIS 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, and geocoding) as well as supports enterprise-level application development based on ArcObjects for the server.

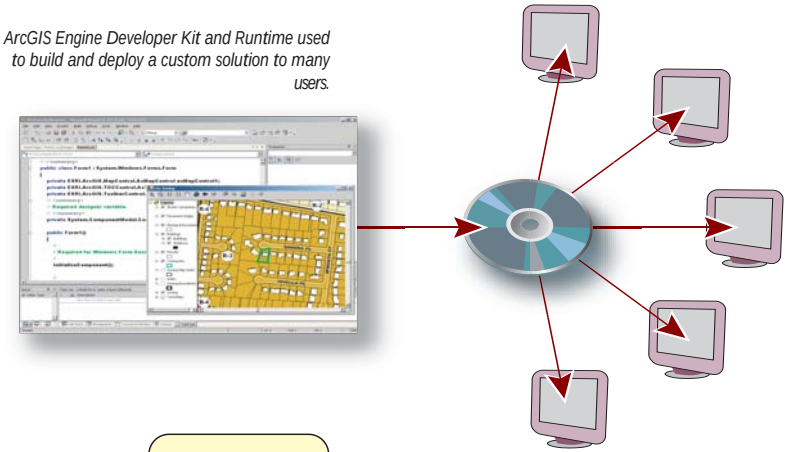
The ArcGIS Server developer kit enables developers to build central GIS servers to host GIS functions that are accessed by many users, perform back office processing on large central GIS databases, build and deliver GIS Web applications, and perform distributed GIS computing.

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 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 to deploy GIS to broad groups of users.

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

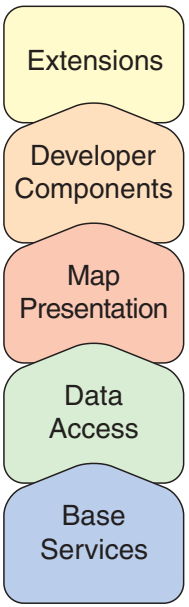
ArcGIS Engine is a complete library of embeddable GIS components for developers to build custom applications. Using ArcGIS Engine, you can embed GIS functions into existing applications, including Microsoft® Office products, such as Word and Excel, and build focused custom applications that deliver advanced GIS systems to many users. ArcGIS Engine consists of a software developer kit and a redistributable runtime providing the platform for all ArcGIS applications. Since ArcGIS Engine is supported on Windows, Solaris, and Linux (Intel), developers can create cross-platform custom solutions for a wide range of users.

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. Extensions—ArcGIS Engine Runtime is deployable with the standard functionality or with additional extensions for advanced functionality.



Components of ArcGIS Engine

Each of these parts, including the extension functionality, is made available through the ArcGIS Engine Developer Kit. The ArcGIS Engine Runtime and its extensions, although integral factors in the development of a custom GIS application, specifically involve application deployment and are, therefore, considered separately.

ARCGIS ENGINE 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 inter-

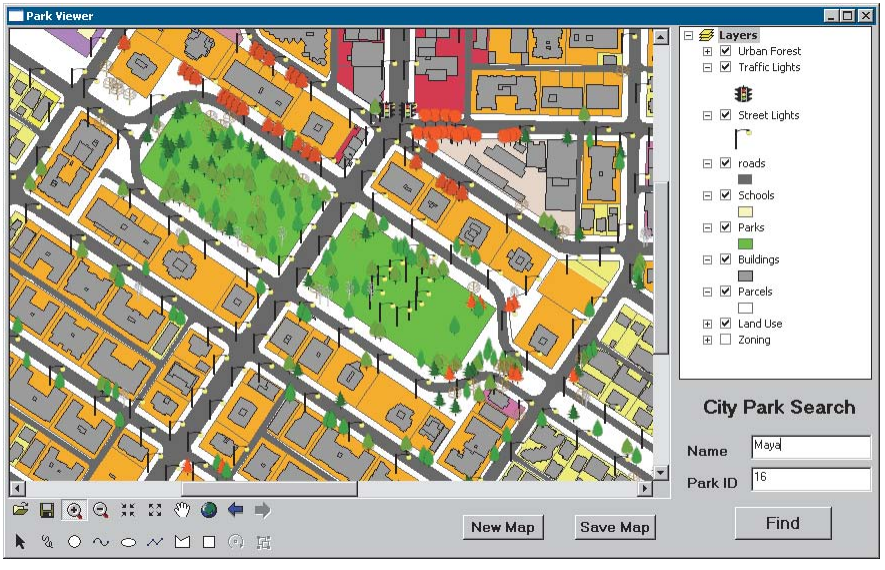
faces for maps. You can use one of several supported APIs to create unique applications or combine ArcGIS Engine components with other software components to realize a synergistic relationship between maps and the information that users manage.

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. Another 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 ArcGIS developer controls are available with each supported API and platform. 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.

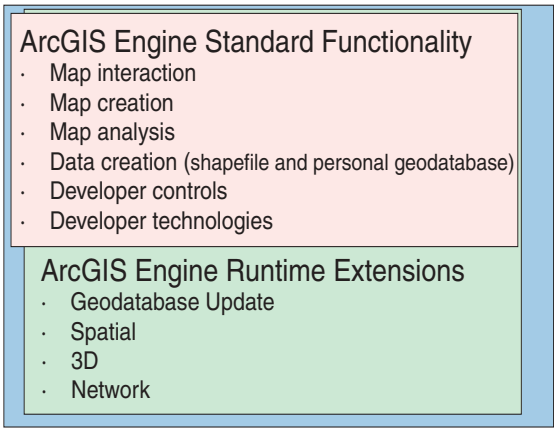
- *MapControl*
- *PageLayoutControl*
- *SceneControl*
- *GlobeControl*
- *ToolbarControl*
- *TOCControl*
- *ReaderControl*
- Collection of commands, tools, and menus for use with the *ToolbarControl*



An ArcGIS controls-based application

ARCGIS ENGINE RUNTIME

The final component of ArcGIS Engine is its extensions. All applications built with the ArcGIS Engine Developer Kit require ArcGIS Engine Runtime, with the appropriate license, to execute successfully. ArcGIS Engine Runtime is the platform on which ArcGIS Desktop is built; this allows users of ArcGIS Desktop applications to execute custom applications based on ArcGIS Engine, if permitted by the ArcGIS Engine application developer. There are several ArcGIS Engine extensions ranging from standard to enterprise extensions.



ArcGIS Engine Runtime deployment options

Standard ArcGIS Engine functionality

The standard ArcGIS 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 data creation, along with the ability to explore features by performing a wide range of spatial or attribute searches. This level also allows basic data creation, editing of shapefiles and simple personal geodatabases, and GIS analysis.

Geodatabase Update extension

The Geodatabase Update extension for ArcGIS Engine Runtime adds the ability to create and update a multiuser enterprise geodatabase managed with ArcSDE. This includes the ability to work with schemas and versioned geodatabases. The Geodatabase Update extension unlocks ArcGIS Engine Runtime with the necessary ArcObjects to run custom editing and advanced geodatabase solutions. These solutions include applications that deal with GIS data automation and compilation and the construction and maintenance of geodatabase features. The Geodatabase Update extension 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 end users that have the ArcGIS Engine Runtime with the Geodatabase Update extension 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 extensions

Three additional extensions are available for the ArcGIS Engine Runtime:

1. Spatial extension—The ArcGIS Engine Runtime Spatial extension provides a powerful set of functions that allow applications to create, query, and analyze cell-based raster data. This type of analysis allows your users 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 extension supports include the calculation of slope, aspect, and contours against digital elevation models (DEMs).
2. 3D extension—The 3D extension for ArcGIS Engine Runtime enables the visualization of data in 3D. This extension supplements standard ArcGIS

Engine with the components for viewing a surface from multiple viewpoints and determining what is visible from a chosen location. *SceneControl* and *GlobeControl* provide the interface for viewing multiple layers of 3D and global data for visualizing data, creating surfaces, and analyzing surfaces.

The StreetMap USA extension functionality is no longer a separate extension for ArcGIS Engine but is included as part of the standard ArcGIS Engine Runtime. The StreetMap USA functions provide street-level mapping, address matching, and basic routing for the USA. StreetMap layers automatically manage, label, and draw features, such as local landmarks, streets, parks, water bodies, and other features, resulting in a rich cartographic street network for the USA.

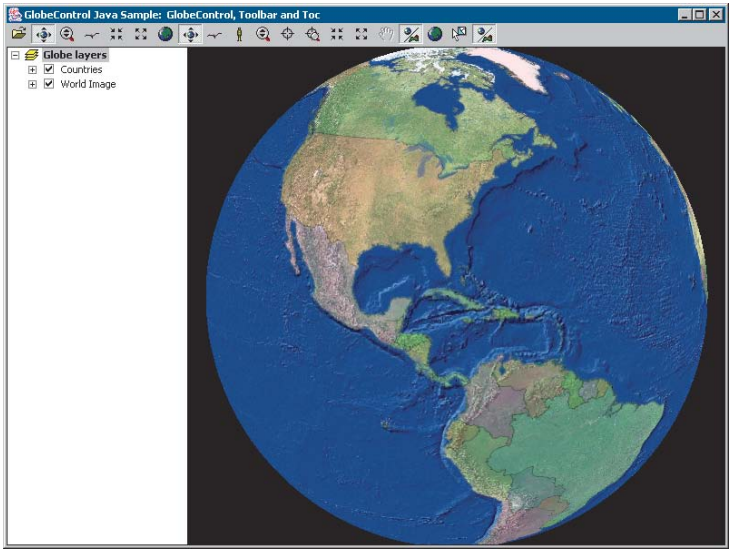
3. Network Analyst extension—The Network Analyst extension is new at version 9.1 and enhances the standard ArcGIS Engine Runtime by adding the capability of routing, service area analysis, and creating and managing network datasets. The Network extension allows developers to create and deploy powerful custom applications for transportation, emergency response, fire, military, and a host of other purposes.

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 in which users need focused, customized access to GIS, ArcGIS Engine provides a lower-cost, lightweight option.

STANDALONE 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. 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 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 standalone 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

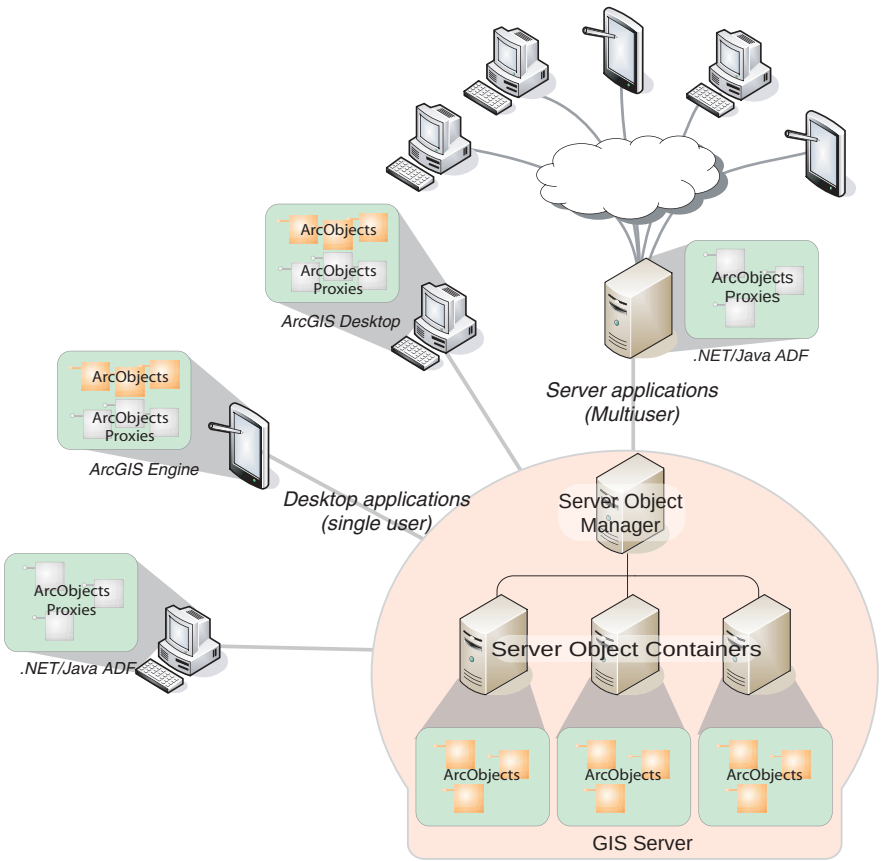
ArcMap, one of the ArcGIS Desktop applications, is an excellent way to create data and author maps for use in custom applications. The *MapControl* and *PageLayoutControl* provided with ArcGIS Engine can work with the map documents created in ArcMap. *SceneControl* and *GlobeControl* can display documents

authored in the ArcScene™ and ArcGlobe™ applications. Using the ArcGIS Desktop applications to create and manage maps used in custom applications can save you much development time 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 applications based on ArcGIS Engine and deploy them 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. It is also important to remember that the GIS functionality exposed via the ArcObjects that compose ArcGIS Engine is the same in ArcGIS Server, meaning that ArcGIS Server, Engine, and Desktop have the same core ArcObjects.



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 extensions.

- Display a map with multiple map layers, such as roads, 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, unions, or inverse intersections of shapes.
- Manipulate the shape or rotation of a map.
- Create and update geographic features and their attributes.

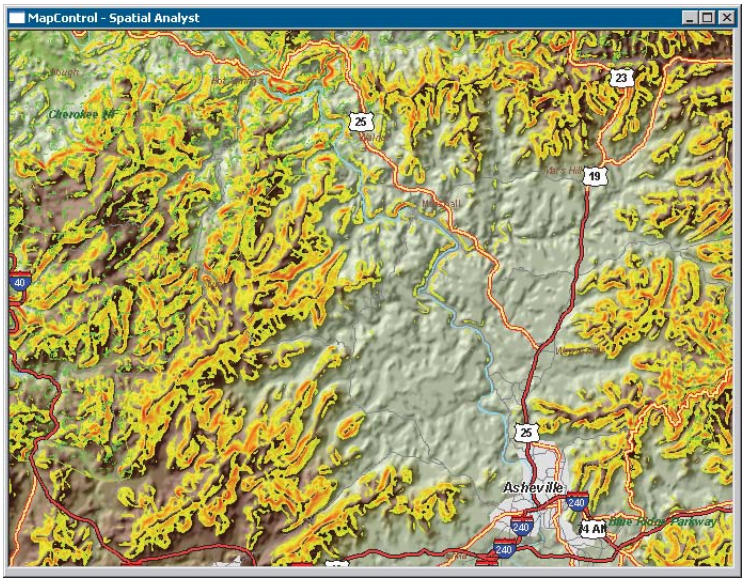
A software authorization file controls the availability of the various levels of ArcGIS Engine Runtime functionality. For more details on deploying and configuring the ArcGIS Engine Runtime, refer to Chapter 5,'Licensing and deployment'.

EDITING FEATURES

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, leveraging the full function of the enterprise geodatabase, the Geodatabase Update extension of the ArcGIS Engine Runtime is required.

SPATIAL MODELING AND ANALYSIS

You can extend the capabilities of ArcGIS Engine by adding the Spatial extension to ArcGIS Engine Runtime. This extension provides a broad range of powerful spatial modeling and analysis functions. You can create, query, map, and analyze cell-based raster data; perform integrated raster or 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 extension for the ArcGIS Engine Runtime*

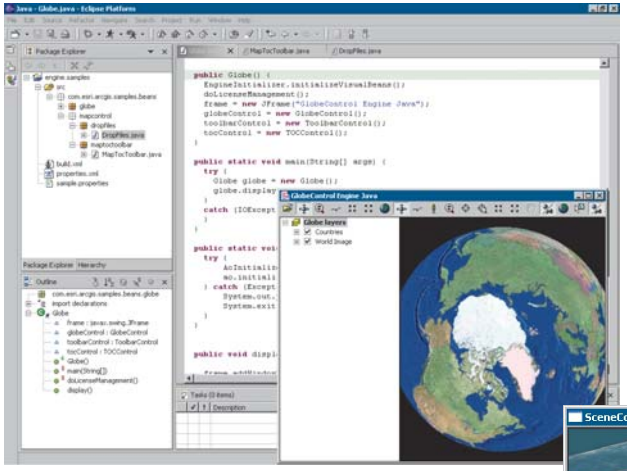
For example, you can:

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

3D VISUALIZATION AND MORE

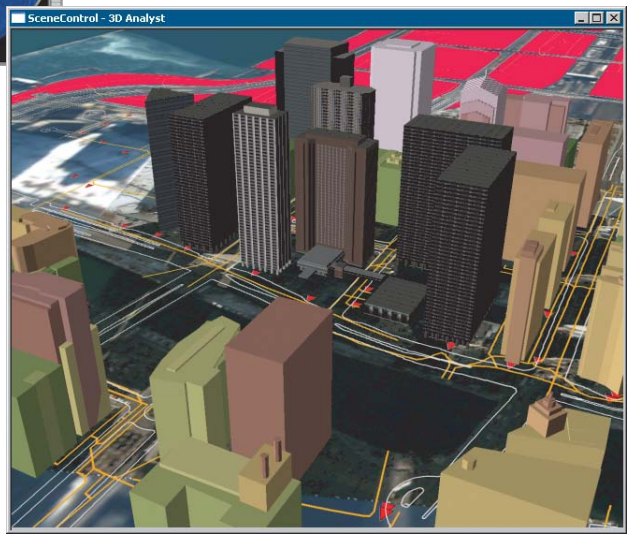
The ArcGIS Engine Runtime 3D extension extends the capabilities of ArcGIS Engine even further by enabling you to build applications that effectively visualize and analyze surface and globe data using *SceneControl* and *GlobeControl*. You can create applications that view a surface from multiple viewpoints, 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.

Java code for the inset GlobeControl*-based application*



You can, for example:

- Display ArcScene and ArcGlobe documents.
- 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.

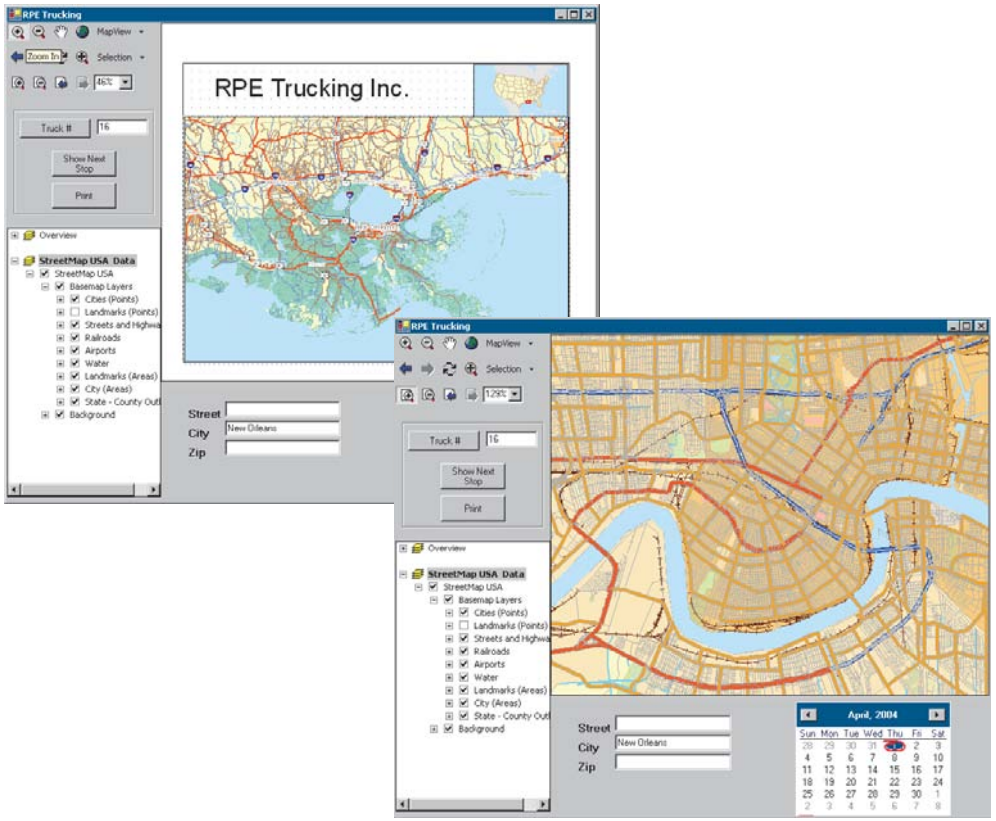


Display of a SceneControl*-based application*

NETWORK ANALYSIS

The Network extension to ArcGIS Engine is new at version 9.1 and provides developers with the capability to create applications that utilize network data in a variety of formats and creating and editing network datasets. The following network functions are available to developers:

- Path—Find a path through a set of network locations that minimizes some impedance (cost) attribute
- Tour—Determine the minimum-cost path to reach a series of stops; also determines the order in which the stops are visited.
- Directions—Generate a series of directions for the user.
- Closest Facility—Given a network location (an incident) finds the closest facilities.
- Service Areas—Find all network elements within a given distance from a network location.



Once you have the ArcGIS Engine Developer Kit installed, you will need to register your product before you can start developing custom applications. At the end of the installation of the ArcGIS Engine Developer Kit, the Software Authorization wizard will start. Follow the steps through the wizard to authorize ArcGIS Engine Developer Kit. The ESRI Customer Service Web site (*http://service.esri.com*) can also be used to obtain your authorization file. To use the Web site or the wizard, you will need to know your product registration number. With the ArcGIS Engine Developer installed and authorized for use, you are ready to get started. 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 discussions and checklists in this section. They are provided to help you formulate your plans and ensure you're getting started on the right foot.

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

DETERMINING THE TYPE OF APPLICATION

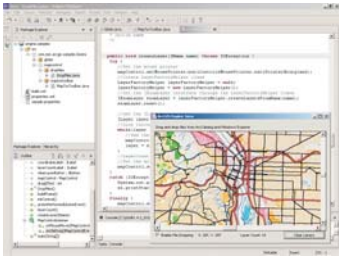
A wide variety of applications can be developed with ArcGIS Engine. These applications vary from simple consoles that perform operations, such as database editing and analyses, to more complex Windows applications that contain controls and visual components for user interaction and geographic data display.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

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

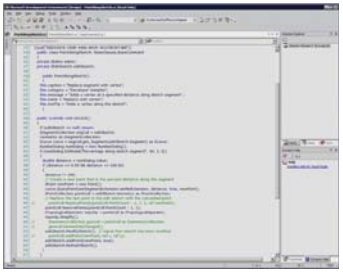
Ultimately, the type of application you develop will depend on the functional requirements of the project at hand.

Checklist:

- ☐ What type of application are you developing? Nonvisual, 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 in the future? Windows? Linux®? Both?



WebSphere Studio



Visual Studio .NET

CHOOSING AN API AND DEVELOPMENT ENVIRONMENT

Since ArcGIS Engine Developer Kit provides four developer APIs—COM, .NET, Java, and C++. The different APIs can be leveraged in several different supported development environments. ESRI recommends and supports the following integrated development environments (IDEs) or compilers when working with ArcGIS Engine.

COM

- Visual Basic 6 sp3 or later
- Visual C++ 6 sp3 or later

- Visual C++ (Visual Studio .NET 2003)

.NET

- C# (Visual Studio .NET 2003 with .NET Framework 1.1)
- VB.NET (Visual Studio .NET 2003 with .NET Framework 1.1)

Java

- Eclipse v. 3.0 or 3.0.1
- JBuilder™ X
- NetBeans 3.6

C++ (Compilers)

- Visual C++ sp3 or later for Windows
- Visual C++ (VS.NET 2003) for Windows
- GCC 3.2 C++ for Linux (Intel)
- WorkShop 6 Update 2 for Sun Solaris

The environment you choose to develop with will ultimately depend on your programming skills, the functionality you wish to provide end users, and whether or not you are integrating with other existing applications or technologies.

Throughout most of this book, VB6 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.

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 want to undertake?

DEVELOPING YOUR APPLICATION

At this point, assuming that a proper project development plan is in place, you are ready to dive into the ArcGIS Engine Developer Kit and start developing your application. You may want to start by identifying the libraries and objects that will be necessary to provide 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 extensions required?
- ☐ How do you plan to deploy the application?
- ☐ Have you implemented the correct license check-out code?

DEPLOYING 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 users already have ArcGIS Engine Runtime installed? No ESRI products installed?
- ☐ What ArcGIS license will your end users have on their systems? ArcInfo, ArcEditor, or ArcView? Which license will your application check for and use?
- ☐ How should you package and deploy the application?
- ☐ Will you need to provide new versions in the future?
- ☐ How will you distribute the application?

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

This book, *ArcGIS Engine Developer Guide*, is an introduction for developers who want to build standalone GIS applications. This guide will help you, as the developer, become familiar with the ArcGIS Engine object model by introducing all the ArcGIS Engine developer kit components, discussing relevant aspects of building applications, introducing supported APIs, and providing developer scenarios that 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-specific 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 application usages of each particular supported API.

CHAPTER GUIDE

Chapter 1, 'Introducing ArcGIS Engine', gives developers an overview of the ArcGIS Engine product, its 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 guides you through each API from the basics to advanced usage topics.

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

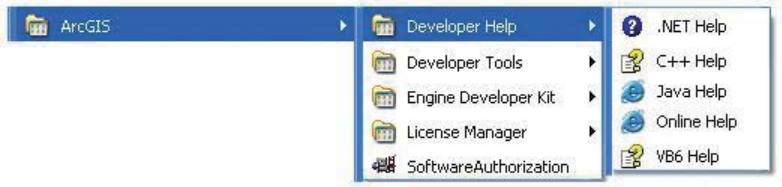
Chapter 6, 'Developer scenarios', guides you through the creation and deployment of several types of standalone applications utilizing each of the supported APIs.

This book also contains a number of appendixes 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.

The ESRI Developer Network at
http://edn.esri.com

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

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

ESRI DEVELOPER NETWORK ONLINE

ESRI Developer Network (EDN) online—*http://edn.esri.com*—is a central location to find and view developer-focused articles, code samples, and other resources related to ArcGIS, ArcIMS, ArcSDE, and ArcWeb Services. In addition, EDN provides a number of community-building areas intended to promote your collaboration and interaction with other GIS developers and ESRI staff. 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 click the Training and Events tab.

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

2 ArcGIS software architecture

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.

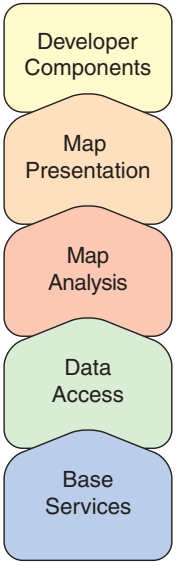
The ArcGIS software architecture supports a number of products, each with its unique set of requirements. ArcObjects, the components that make up ArcGIS, are designed and built to support this. This chapter introduces ArcObjects.

ArcObjects is a set of platform-independent software components, written in C++, that provides 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'.

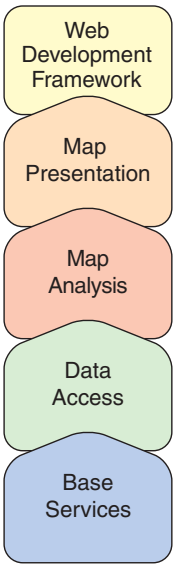
As stated, the language chosen to develop ArcObjects was C++; in addition to this language, ArcObjects makes use of the Microsoft Component Object Model. COM is often thought of as simply specifying how objects are implemented and built in memory and how these objects communicate with one another. While 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 equally. 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:

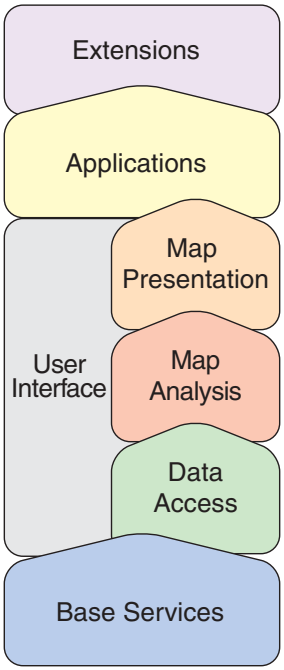


ArcGIS Engine

- ArcGIS Engine—Use of the object is within a custom application. Objects within ArcGIS Engine must support a variety of uses; simple map dialog boxes, multithreaded servers, and complex Windows desktop applications are all possible uses of ArcGIS Engine objects. The dependencies of the objects within ArcGIS 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 ArcGIS 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 Server



ArcGIS Desktop

Many of the ArcObjects components 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 of 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 among 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 among 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. Some of the changes from ArcGIS 8 to ArcGIS 9 may 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 and multithreaded server applications.
- Multiple Platform Support—ArcObjects for ArcGIS Engine and Server should be capable of running on multiple computing platforms.
- Compatibility—ArcObjects 9 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 ArcObjects components 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 Dynamic Link Libraries (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 those 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 criterion, thought is given to the end use and the modularity required for support that. For example, the system could be divided into many small DLLs with only a few objects in each. Although 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.

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 requirements of being thread safe, scalable, without 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 has user interface components and is a Windows-only library.

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.

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.

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 that the resulting application will provide vastly improved performance.

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

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.

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.

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.

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.

Although 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 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 Object Linking and Embedding Database (OLE DB).

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

COMPATIBILITY

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

Although the changes required for 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 ArcGIS Engine, ArcGIS Server, and ArcGIS Desktop.

ESRI understands the importance of a unified software architecture and has made numerous changes for ArcGIS 9 so 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 about 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. 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. ArcGIS Engine supports the following APIs:

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 (for example, Visual Basic and Visual C++) can be used with this API.
- .NET—Visual Basic .NET and C# are supported by this API.
- Java—Sun™ Java 2 Platform, Standard Edition (J2SE).
- C++—Microsoft Visual C++ 6.0, Microsoft Visual C++ .NET 2003, Sun Solaris Forte 6 Update 2, Linux GCC 3.2.

When working with ArcObjects, developers can consume functionality exposed by 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 is developed in C++, there are some cases in which data types compatible with C++ have been used for performance reasons. These performance considerations mostly affect the internals of ArcObjects; therefore, using one of the generic interfaces should not adversely affect performance of your ArcObjects developments.

All four APIs support consuming the functionality of 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" suffix 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 nongeneric interface is supported by the API, it is possible to continue to use the API-specific interface.

EXTENDING API

Extending ArcObjects entails 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 among the APIs and, in some cases, varies among languages of an API.

The COM API provides the most possibilities for extending the system. The limitation within this API is with 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*. For example, *ICurve*, which inherits from *IGeometry*, cannot be implemented in VB for this reason.
- Method names on an interface start 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 reuse 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; the one exception is interfaces that make use of data types that are not compliant with OLE automation. See the table below for a complete list of OLE automation-compliant data types.

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

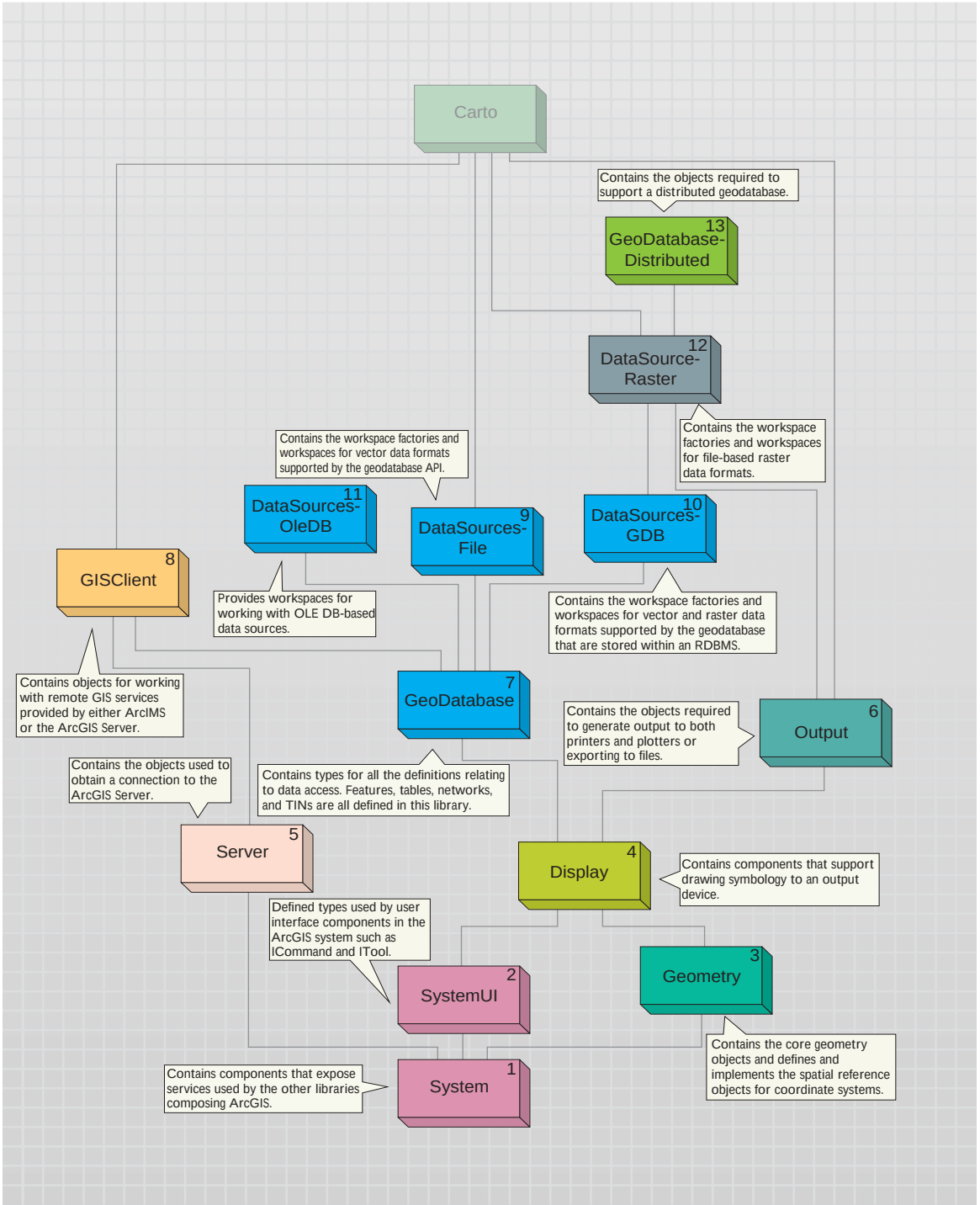
Type	Description
Boolean	Data item that can have the value True or False.
unsigned char	8-bit unsigned data item.
double	64-bit IEEE floating-point number.
float	32-bit IEEE floating-point number.
int	Signed integer, whose size is system dependent.
long	32-bit signed integer.
short	16-bit signed integer.
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 table of contents (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 four ArcGIS Engine APIs.

The COM and .NET APIs are only supported on the Microsoft Windows plat-

form, while the Java and C++ APIs are supported on all the platforms supported by ArcGIS Engine.


Carto
Contains the objects required to support a distributed geodatabase.
13
GeoDatabase-Distributed
12
DataSource-Raster
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.
11
DataSources-OleDB
9
DataSources-File
10
DataSources-GDB
8
GISClient
Provides workspaces for working with OLE DB-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.
7
GeoDatabase
Contains the objects required to generate output to both printers and plotters or exporting to files.
6
Output
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.
5
Server
4
Display
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.
2
SystemUI
3
Geometry
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.
1
System

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 ArcGIS Engine are summarized below. The diagrams that accompany this section indicate the library architecture of ArcGIS Engine. Understanding the library structure, 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 precede 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 the system library that can be implemented by the developer. The *AoInitializer* object is defined in System; all developers must use this object to initialize and uninitialize ArcGIS Engine in applications that make use of ArcGIS 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. For 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 ArcGIS Engine. These include the *ICommand*, *ITool*, and *IToolControl* interfaces. The developer uses these interfaces to extend the UI components that ArcGIS Engine 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 with which most users will interact 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*, *Path*, 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 endpoint, and an element type that defines the curve from start to end. The kinds of segments are *CircularArc*, *Line*, *EllipticArc*, and *BézierCurve*. 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 in-

cluded 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, and so forth). 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, and so forth); in addition, it can 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, computer-aided design (CAD), Spatial Data Compressed (SDC), and vector product format (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

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

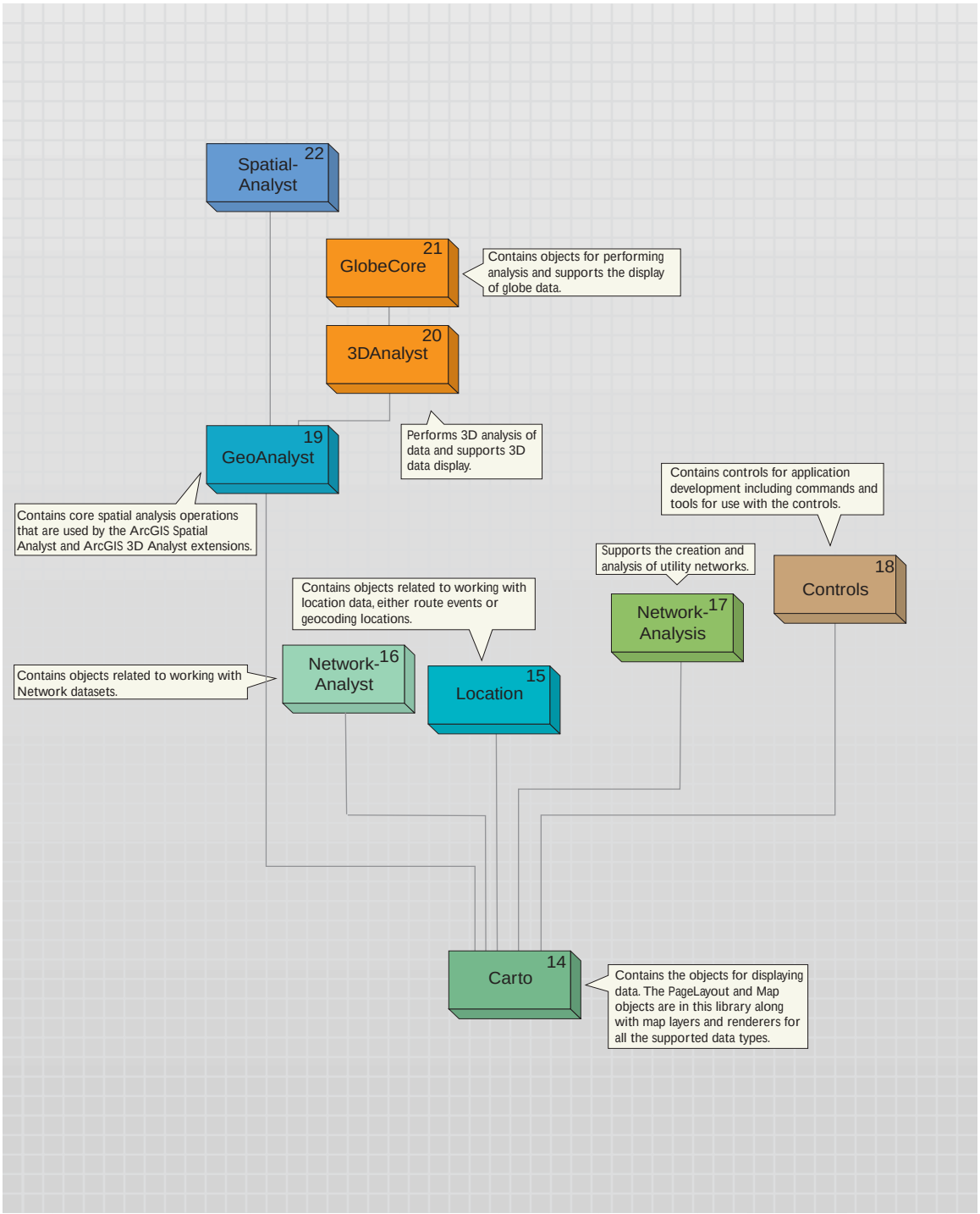
The DataSourcesRaster library contains the implementation of the GeoDatabase API for the raster data sources. These data sources include relational database management systems supported by ArcSDE—IBM DB2, Informix, Microsoft SQL Server, and Oracle—along with supported Raster Data Objects (RDO) raster 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.

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, scalebars, and so forth. The *Map* object is a container of layers. The *Map* object has properties that operate on all


Spatial-Analyst 22
GlobeCore 21
Contains objects for performing analysis and supports the display of globe data.
3DAnalyst 20
Performs 3D analysis of data and supports 3D data display.
GeoAnalyst 19
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.
Controls 18
Network-Analysis 17
Contains objects related to working with location data, either route events or geocoding locations.
Contains objects related to working with Network datasets.
Network-Analyst 16
Location 15
Carto 14
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.

The ArcGIS Server uses the MapServer *object for its Map Service.*

layers within the map—spatial reference, map scale, and so forth—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 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.

Although developers can directly make use of the *Map* or *PageLayout* objects in their applications, it is 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, and so forth, 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 offer 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.

NETWORKANALYST

The NetworkAnalyst library contains objects for working with network datasets. Developers can extend this library by creating new network servers. A license for the Network Analyst extension of the ArcGIS Engine Runtime Network option is required to make use of the objects in this library.

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, and so forth.

CONTROLS

The contents of the Map *and* PageLayout *controls can be specified programmatically, or they can load map documents.*

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.

The ReaderControl *only supports Published Map Files.*

ArcGIS Engine comes with more than 150 commands.

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 ArcGIS SpatialAnalyst and ArcGIS 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 extension 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 extension is required to work with objects in this library.

GLOBECORE

The GlobeCore library contains objects for working with globe data similar to the 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 extension 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 extension 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, ToolbarControl, and LicenseControl. The GlobeControl and SceneControl are also available, but applications using these controls must be authorized with the ArcGIS Engine 3D extension.

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.

ArcGIS controls are high-level developer components that enable you to build and extend applications with ArcGIS functionality and provide a graphical user interface. They simplify your development process by encapsulating ArcObjects and providing a coarser-grained API. The controls allow you to easily deploy well-crafted applications with a common look and feel.

Each of the controls provided with ArcGIS Engine is available as an ActiveX® Control, Motif widget, .NET Windows control, and visual JavaBean™. They include:

The TOC in TOCControl stands for Table of Contents.

A valid ArcGIS Engine Developer Kit license enables you to create applications with all of these controls; however deployment of applications built with either GlobeControl *or* SceneControl *requires both a core ArcGIS license—Engine Runtime or Desktop—and its corresponding 3D extension license. For more information on deployment of ArcGIS controls-based applications, see Chapter 5,'Licensing and deployment'.*

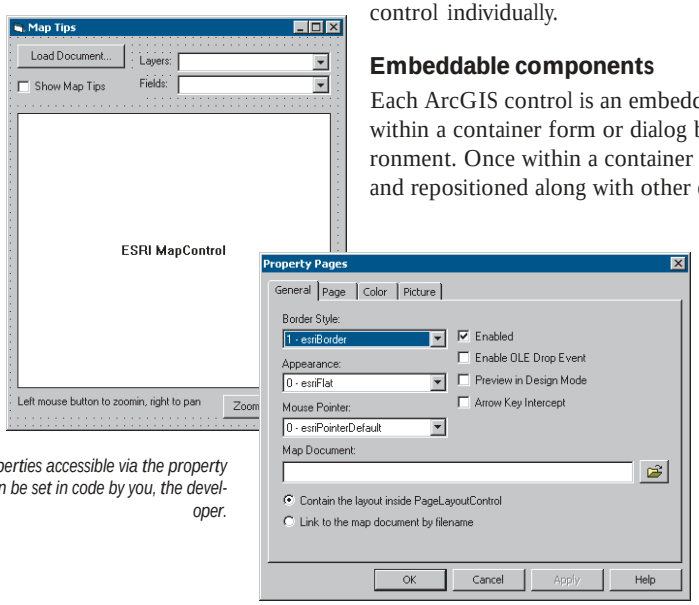
- *MapControl*
- *PageLayoutControl*
- *ReaderControl*
- *ToolbarControl*
- *TOCControl*
- *LicenseControl*
- *SceneControl*
- *GlobeControl*

For information on developing with the ArcReaderControl*, see the Publisher library overview topic in your ArcGIS Desktop Developer Kit's help system.*

ArcGIS Desktop developers may also be familiar with the *ArcReaderControl*. This control is available to ArcGIS Desktop developers who also have the required ArcGIS Publisher extension license. This control is not included with ArcGIS Engine and therefore is not discussed in this book.

ABOUT THE CONTROLS

This topic discusses methodology and concepts applicable to all of the ArcGIS controls. Later topics in this section discuss specific aspects of each control individually.



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

Embeddable components

Each ArcGIS control is an embeddable component that can be dropped within a container form or dialog box 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.

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 the control and clicking Properties from the context menu. These property pages provide shortcuts to a selection of a control's properties and

methods and allow you, as 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* may 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 because they provide not only a user interface but also 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.

For more details on how the controls work together, see the 'TOCControl and ToolbarControl' section later in this chapter.

Buddy Controls

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 through the control property pages (in development environments that support property page capability) or programmatically using SetBuddyControl.

Map authoring

The ArcGIS Desktop applications can be used to preauthor 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*. Preauthoring documents can substantially reduce your development time as it saves having to programmatically build up maps and symbology from scratch. Once a document is loaded into an ArcGIS control, any layer, element, and symbol can be accessed programmatically through the object model if its appearance subsequently needs changing.

You can also save the contents of your controls to map documents using the *MapDocument* class with the *MapControl* or *PageLayoutControl*. The documents can then be reopened by the control or by ArcMap. Using this technique it is possible to share documents between your custom application and ArcGIS Desktop. For

more information on authoring map documents, see the ArcGIS Desktop Help system.

The table below summarizes the types of documents that can be loaded into each ArcGIS control.

	Map Document (.mxd, .mxt)	Layer Files (.lyr)	Scene Document (.sxd, .sxt)	Globe Document (.3dd, .sdt)	Published Map Files (.pmf)		
					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 to the* ReaderControl

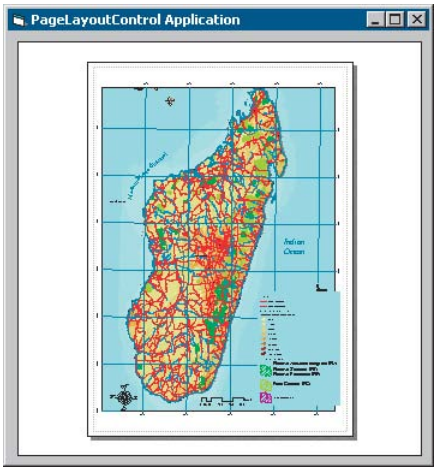
*** There are no properties available on the ArcGIS controls to directly load Layer (.lyr) files. However, they can be loaded indirectly via the* MapDocument *object.*

Application built using the MapControl

MAPCONTROL AND PAGELAYOUTCONTROL

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 avoid 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 onward. Alternatively, a map document can be loaded into



Application built using the PageLayoutControl

ArcGIS Desktop developers may already be familiar with MapControl *and* PageLayoutControl *since these controls were initially made available in previous ArcGIS Desktop releases. With the release of ArcGIS Engine, these controls have been incorporated into its package of developer components. As in previous releases, ArcGIS Desktop developers can build applications with* MapControl *or* PageLayoutControl *even without an ArcGIS Engine license. However, they cannot work with the additional controls provided with ArcGIS Engine unless you have an Engine Developer Kit license.*

the control programmatically using the *LoadMxFile* method.

Not only can the *MapControl* and *PageLayoutControl* read map documents, they can also write map documents. 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.

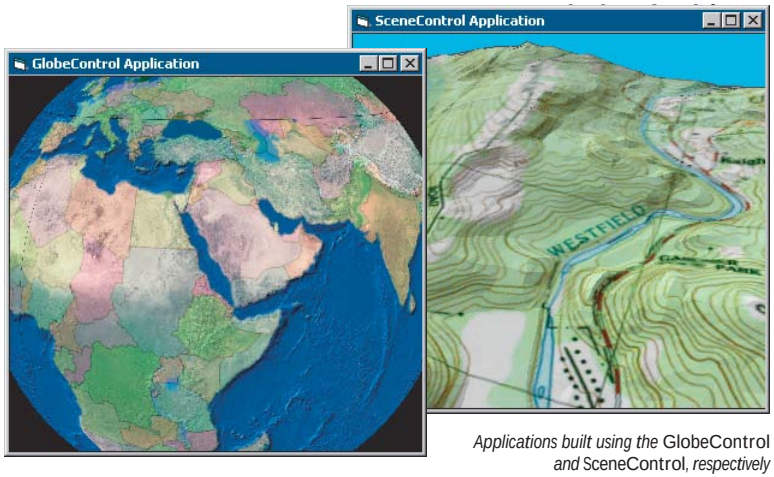
Helper methods, such as *TrackRectangle*, *TrackPolygon*, *TrackLine*, and *TrackCircle*, exist on the *MapControl* for tracking or "rubberbanding" shapes on the display. The *VisibleRegion* property can be used to change the shape of the *MapControl*'s display area. Helper methods, such as *FindElementByName*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

GLOBECONTROL AND SCENECONTROL

Deployment of applications built with either GlobeControl *or* SceneControl *requires both a core ArcGIS license—Engine Runtime or Desktop—and its corresponding 3D extension license. For more information on deployment of ArcGIS controls-based applications, see Chapter 5,'Licensing and deployment'.*

The *GlobeControl* and *SceneControl* correspond to the 3D views of the ArcGlobe and ArcScene 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 avoid 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

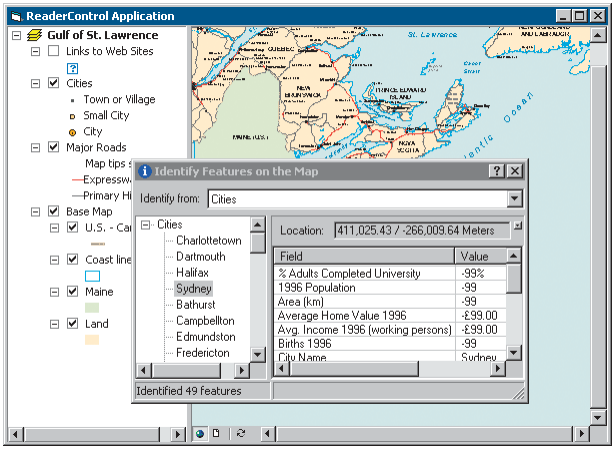


Applications built using the GlobeControl *and* SceneControl, *respectively*

READERCONTROL

ArcReader is available for download. To get your free copy, go to http://www.esri.com/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 application, such as the Find window and the Identify tool. Published Map Files (PMF) authored with ArcMap and published with the ArcGIS Publisher extension can be loaded into the *ReaderControl*, if published with permission to load into a customized ArcReader application.



A ReaderControl *application*

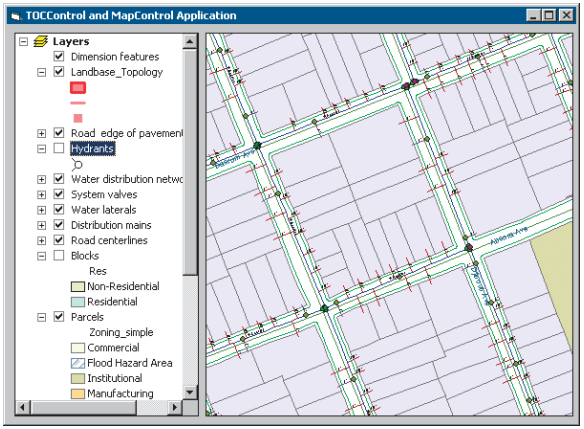
The *ReaderControl* has a simple self-contained object model that exposes all the functionality of the ArcReader application and does not require access to ArcObjects. As such, developing applications with the *ReaderControl* does not require previous experience with ArcObjects. However, if a Published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

For more information on the ArcReaderControl*, see the Publisher library overview topic in your ArcGIS Desktop Developer Kit's help system.*

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 you access any underlying ArcObjects components. Developing with the *ArcReaderControl* requires the ArcGIS Publisher extension, and applications built with the *ArcReaderControl* can be deployed on any machine that has the free ArcReader 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 through 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.



A TOCControl *application*

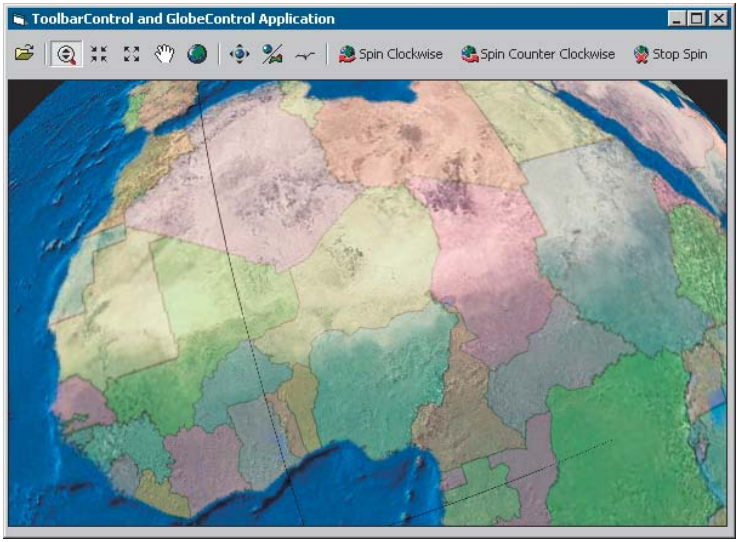
To link a custom control that you have created to the TOCControl *the custom control must implement the* ITOCBuddy *interface.*

Each *TOCControl* buddy control implements the *ITOCBuddy* interface. The *TOCControl* works with 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 *MapControl*, the map layer will also be removed from the *TOCControl*. Likewise, if the end user interacts with the *TOCControl* to uncheck a map layer's visibility, the layer will no longer be visible within the *MapControl*.

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 through 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.

If you are using JavaBeans, the buddy control can only be set through code as this property is not exposed in the integrated development environment (IDE) property pages.



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.

To link a custom control that you have created to the ToolbarControl *the custom control must implement the* IToolbarBuddy *interface.*

CONTROL COMMANDS

ArcGIS Engine provides a set of commands, tools, and menus to work with the ArcGIS controls. For example, there is a map navigation, feature selection, and graphic element commands suite that works 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, programmatically, or at runtime by the end user if the *ToolbarControl* is in customize mode.

You can also extend the suite of commands provided by 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' scenarios in Chapter 6, 'Developer scenarios', to see how to build a custom command using the *HookHelper* object.

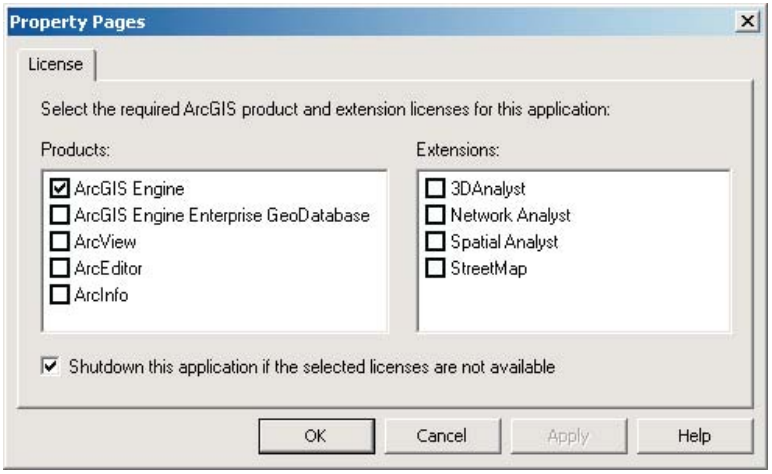
LICENSECONTROL

The LicenseControl is only available with the COM (ActiveX Control) and .NET (Windows Control) APIs.

The LicenseControl is used to initialize an application with a suitable license(s) for it to run successfully on any machine it is deployed on to. The LicenseControl will configure the licenses at application start time when the form or dialog box containing the LicenseControl is loaded. Use the LicenseControl to automatically perform license initialization within simple graphical user interface applications using the MapControl, PageLayoutControl, TOCControl, ToolbarControl, ReaderControl, SceneControl, or GlobeControl. If greater control is required over license initialization, particularly when checking out and in extension licenses (the LicenseControl will check out extension licenses for the duration of an application's life), use the *AoInitialize* object in the System library to programmatically perform license initialization.

Refer to Chapter 5 'Licensing and deployment' for more details about the concepts of licensing.

The LicenseControl is visible within a visual design environment but is invisible at runtime, so unlike the other controls the LicenseControl does not provide any user interface in an application. In design time the LicenseControl property pages must be used to configure the application with product and extension licenses.



Products

Select at least one product license with which the application can be initialized. By default, the LicenseControl will try to initialize the application with the ArcGIS Engine product license. If the product you require is not licensed, you may optionally initialize the application with a higher product license. For example, if you select the ArcGIS Engine license and the ArcView license, the

LicenseControl will initially try to initialize the application with an ArcGIS Engine license (the lower license). If that license is not available, the LicenseControl will try to initialize the application with an ArcView license (the next higher level license selected). If no product licenses are available, then the application will fail to initialize. Note that once an application is initialized with a product license, it is not possible to reinitialize the application for the duration of the application's life.

Extensions

Select the extension licenses required by the application. Not every extension license is available with every product license; as such the list of available extension licenses will change as different product licenses become selected. The availability of each extension license is checked in conjunction with the product license with which the application will ultimately be initialized. If any of the selected extensions are not available, the application will fail to initialize. The LicenseControl will check out extensions directly after the application is initialized and will check in extensions when the application is shutdown.

If a SceneControl or GlobeControl (requiring the 3D Analyst extension) is embedded within the same container as the LicenseControl, the 3D Analyst extension will automatically be checked.

Shutdown

Set whether the LicenseControl will automatically shut down the application if license initialization fails. If the LicenseControl handles license initialization failure, a License Failure dialog box will be displayed to the user before the application is shut down. If the developer handles license initialization failure, the *LicenseAvailability*, *Status*, and *Summary* properties can be used to obtain information on the nature of the failure before the application is programmatically shut down.

For step-by-step scenarios walking you through the ArcGIS controls development process, refer to the 'Building applications' scenario of your choice in Chapter 6, 'Developer scenarios'.

The ArcGIS controls can be used to build applications in two ways: the ArcGIS controls can be embedded into an existing application to add additional mapping capability, or the ArcGIS controls can be used to create a new standalone 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 application's framework. The following sections discuss the application development process for the controls both when you are utilizing the *ToolbarControl* and when you choose not to.

APPLICATION DEVELOPMENT USING THE TOOLBARCONTROL

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 application's framework. ArcGIS Desktop applications, such as ArcMap, ArcGlobe, and ArcScene, have a powerful and flexible framework that includes 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 dialog boxes, forms, and multiple docking interface (MDI) applications. They also provide generic user interface components such as 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. You 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. You 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 with which the command will work. 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.

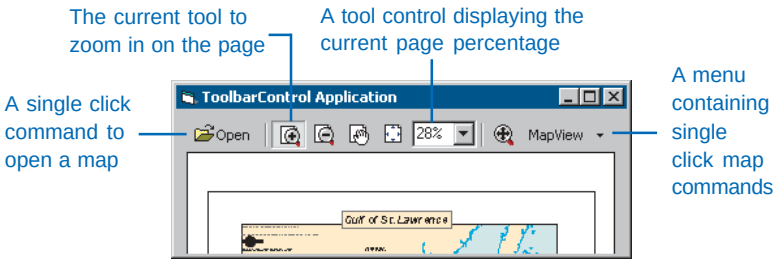
To help you create custom commands to work with the ArcGIS controls and the ArcGIS Desktop applications, *HookHelper*, *GlobeHookHelper*, and *SceneHookHelper* objects exist.

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

Rather than requiring you to add code into a command's *OnCreate* method to determine the type of hook passed to the command, the helper object handles this. The helper objects are used to hold onto 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' scenarios 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 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 commands:

- 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 an 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 commands 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, such as 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*.

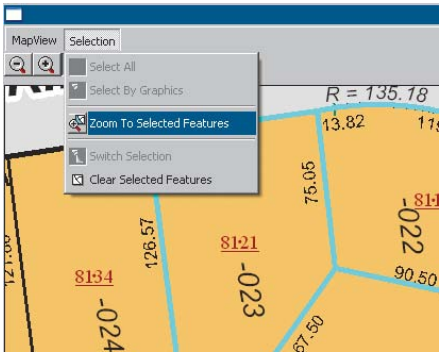
Commands can be added to *ToolbarControl* in two ways: by specifying a *UID* object that uniquely identifies a command (using a Globally Unique Identifier, or GUID) or by supplying an instance of an existing *Command* object to the *AddItem* method. Where possible, commands should be added to the *ToolbarControl* 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 one-half second to ensure that the appearance of each ToolbarItem hosted on the *ToolbarControl* is synchronized 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 you must call the *Update* method programmatically to refresh the state of each ToolbarItem.

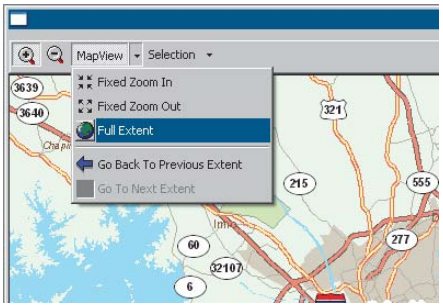


Hosting the ToolbarMenu directly on the ToolbarControl

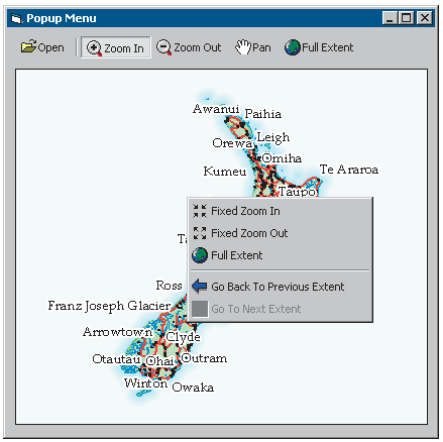
The first time the *Update* method is called in an application, the *ToolbarControl* will check whether the *ICommand::OnCreate* method of each ToolbarItem's underlying command has been called. If the method has not been called, the *ToolbarControl* is automatically passed as the hook to the *ICommand::OnCreate*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

ToolbarMenu

The *ToolbarControl* can host an item that is a dropdown 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 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 hosted on the *ToolbarControl*, be hosted on



Hosting theToolbarMenu as a submenu



Hosting the ToolbarMenu as a popup

another ToolbarMenu as a submenu, or appear as a popup menu and be used for a right-click context menu. Refer to the 'Building applications' scenarios in Chapter 6, 'Developer scenarios', to see how to build a popup menu hosting some control commands that work with the *PageLayoutControl*.

CommandPool

Each *ToolbarControl* and ToolbarMenu has a *CommandPool* that is used to manage the collection of *Command* objects that it is using. Normally, you 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.

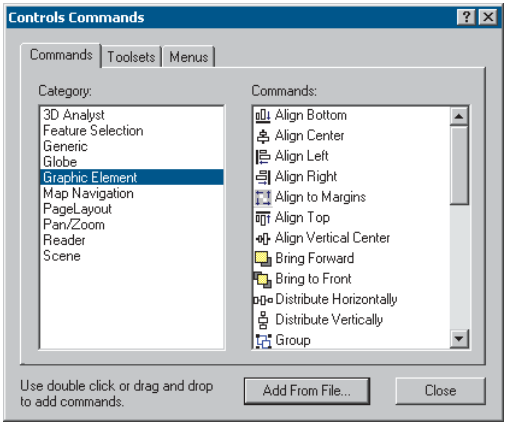
For example, if a Zoom In tool is added to a *ToolbarControl* twice, with the *UID* supplied, when one of the Zoom In items on the *ToolbarControl* is selected and appears "pressed", the other Zoom In item will also appear pressed because they are both using the same *Command* object. When an application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

Customize

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*, then either drag the selected item to a new position or drag and drop the item off the *ToolbarControl* to remove it.
- Right-click 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, you 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



The modeless CustomizeDialog box for ToolbarControl

Toolbars, and ESRI Controls Menus component categories. If required you 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 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. 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' scenarios in Chapter 6, 'Developer scenarios', to see how to display the CustomizeDialog when the *ToolbarControl* is in customize mode.

OperationStack

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 the operation can be rolled forward and 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 on its implementation. Typically, you create 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 proceed through the OperationStack.

APPLICATION DEVELOPMENT WITHOUT THE TOOLBARCONTROL

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, you 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 a command button supplied by the development environment.

Alternatively, the control commands that are provided with ArcGIS Engine, or any custom commands that make use of the *HookHelper*, *SceneHookHelper*, or *GlobeHookHelper* objects, will work directly with an individual ArcGIS control. However, you become 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.
- You can use the *CommandPool* object without the *ToolbarControl* 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, you 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, you 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, 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 want to undertake. At a minimum, review 'The Microsoft Component Object Model' and 'Developing with ArcObjects' sections, 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 would be possible to write a piece of code once and 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 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 is 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 characterizes 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 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 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 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.

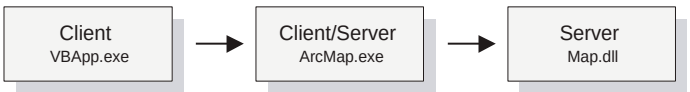
COM allows these objects to be reused at a binary level, meaning that third-party developers do not require access to source code, header files, or object libraries to extend the system, even at the lowest level.

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 these terms.) Therefore, it is worthwhile to define some terminology.

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 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 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.

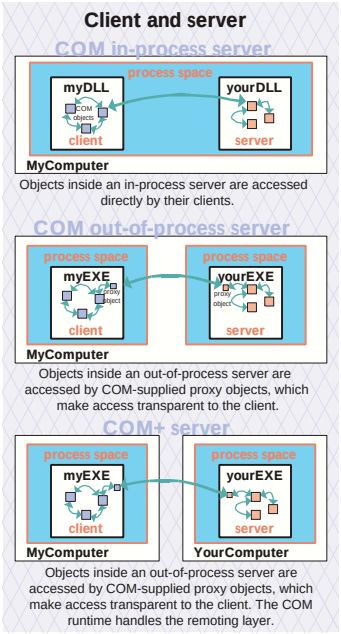
When creating COM objects, the developer must be aware of the type of server that the objects will reside in, but if the creator of the object has implemented them correctly, the packaging does not affect the use of the objects by the client.

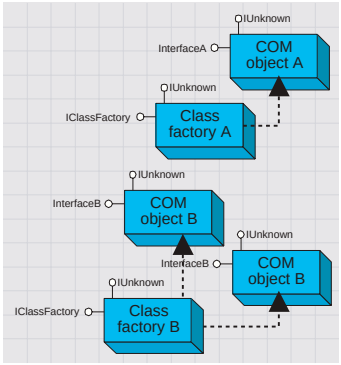
There are pros and cons to each method of packaging that are symmetrically opposite. DLLs are faster to load into memory, and calling a DLL function is faster. EXEs, on the other hand, provide a more robust solution (if the server fails, the client will not crash), and security is better handled since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

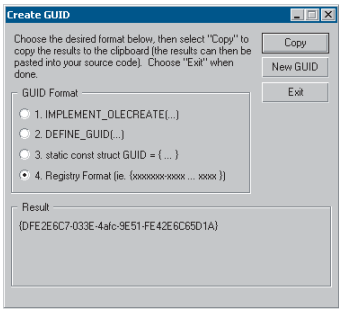
In a COM system, the client, or user of functionality, is completely isolated from the provider of that functionality, the object. All the client needs to know is that the functionality is available; with this knowledge, the client can make method calls to the object and expect the object to honor them. 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.





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.*



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".

CLASS FACTORY

Within each server there is an object called a class factory that the COM runtime interacts with 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

Although 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 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.

GLOBALLY UNIQUE IDENTIFIERS

A distributed system potentially has 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 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.

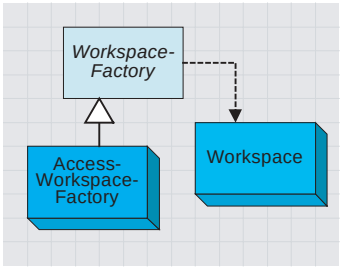
```
{E6BDAA76-4D35-11D0-98BE-00805F7CED21}
```

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 which requests can be made of an object that chooses to implement the interface.

How an interface is implemented differs among 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 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



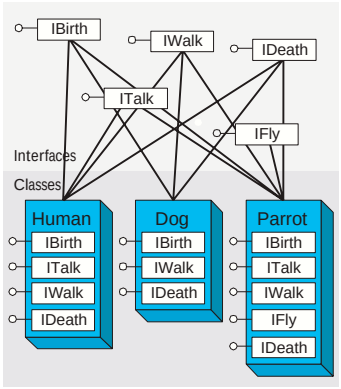
This is a simplified portion of the geodatabase object model showing type inheritance among abstract classes, coclasses, and instantiation of classes.

not a shortcoming since individual classes can implement multiple interfaces. See the diagram to the lower 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 and 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 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 lower 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.



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

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 to 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 nor 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.

Interfaces allow developers to model functionality abstractly. Visual C++ developers see interfaces as collections of pure virtual functions, while Visual Basic developers see interfaces as collections 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:

- An interface is not a class. An interface cannot be instantiated by itself since it carries no implementation.
- 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.

- 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.
- 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 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

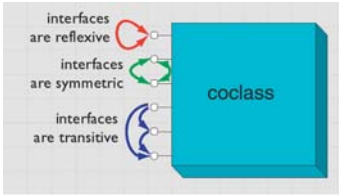
All COM interfaces derive from the *IUnknown* interface, and all COM objects must implement this interface. The *IUnknown* interface performs two tasks: it controls object lifetime and provides runtime type support. It is through the *IUnknown* interface that clients maintain a reference on an object 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* 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 symmetrical, 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, symmetrical, and transitive. It is always possible, holding a valid interface pointer on an object, to get any other interface on that object.*

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

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.*

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

MIDL is commonly referred to as IDL.

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

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* interface, 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.

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.

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.

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 DCE, where it was 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.

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.

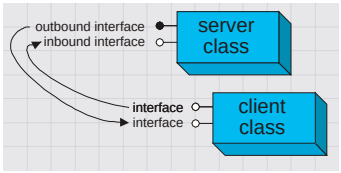
To support the concept of a language-independent development set of components, all relevant data concerning the ArcObjects libraries is shipped inside type libraries. There are no header files, source files, or object files supplied or needed by external developers.

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 in which 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.



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

Dispatch event interfaces

There are some objects within ArcObjects that support two outbound event interfaces that look similar to the methods they support. Examples 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, consult ArcGIS Developer Help for more details on using the interface.

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:

- 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.
- ID. Method IDs are stored at compile time, but execution of the method is still performed through a higher-level function.
- Custom vTable (early). Binding is performed at compile time. The client can then make method calls directly into the object.

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

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.

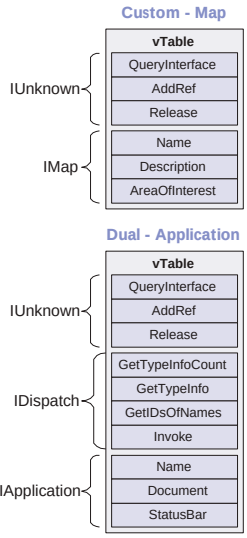
Assuming the required method is supported, the client executes the method by calling the *IDispatch::Invoke* method. This method, in turn, calls the required method and returns the status and any parameters back to the client on completion of the method call.

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.

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.



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.*

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 to execute the required method on the object.

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.

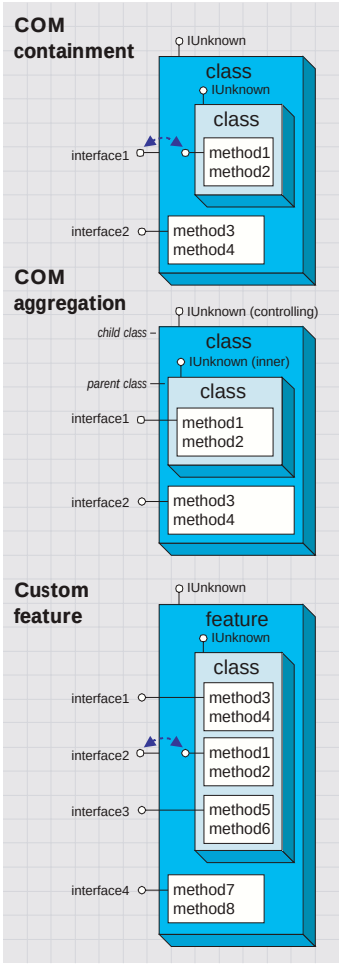
INTERFACE INHERITANCE

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

An interface consists of a group of methods and properties. If one interface inherits from another, then all 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 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.

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

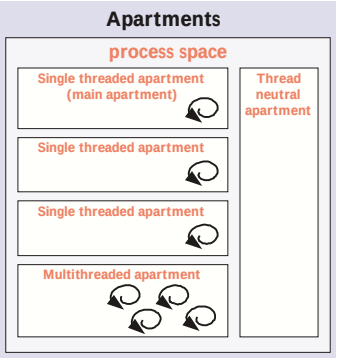
THREADS, APARTMENTS, AND MARSHALLING

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 works 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 apartments and multithreaded apartments (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 apartments 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.

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:



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 with which they work.

- Apartment. Object must be executed within the STA. Normally used by UI objects.
- Free. Object must be executed within the MTA. Objects creating threads are normally placed in the MTA.
- Both. Object is compatible with all apartment types. The object will be created in the same apartment as the creator.
- 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+.

Marshalling enables a client to make interface function calls to objects in other apartments transparently. Marshalling 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 marshaller that handles function calls that use automation-compliant data types (see table below). Nonautomation data types can be handled by the standard marshaller as long as proxy stub code is generated; otherwise, custom marshalling code is required.

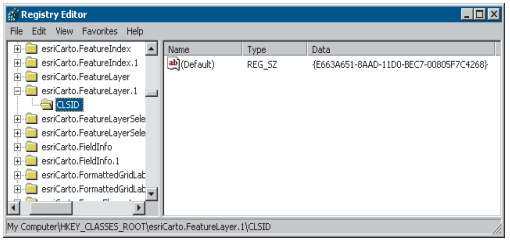
Type	Description
Boolean	Data item that can have the value True or False
unsigned char	8-bit unsigned data item
double	64-bit IEEE floating-point number
float	32-bit IEEE floating-point number
int	Signed integer, whose size is system dependent
long	32-bit signed integer
short	16-bit signed integer
BSTR	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

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 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.



ESRI keys in the Windows system registry

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 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.

ArcGIS applications are built using ArcObjects and can be developed via several APIs. These include COM (VB, VC++, 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.

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.

For simplicity, some samples will not follow the coding standards. For 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 in which a name clash will occur. Omitting this text makes the code easier to understand for developers new to ArcObjects.*

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. Appendix A, 'Reading the object model diagrams', provides a detailed introduction to the class diagrams and shows many of the common routes through 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

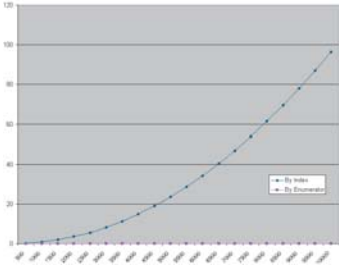
All interfaces and coclasses are documented in the component help file. Ultimately, this will be the help most commonly accessed when you get to know the object models better.

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 the ArcGIS Developer Help system.

Code wizards

There are a number of code generation wizards available to help with the creation of boilerplate code in Visual Basic, Visual C++, and .NET. Although 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.



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

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.

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 exponentially ($y=cx^b$).

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 specific section in this chapter for the language with which you are developing.

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. Suppose you have a polygon, and you want to obtain 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 follows:

```
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. When 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 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 no specific property is stated, so 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 were 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 your initialization routines like this:

```
set ViewEvents = MapControl1.map
```

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 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 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. For 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

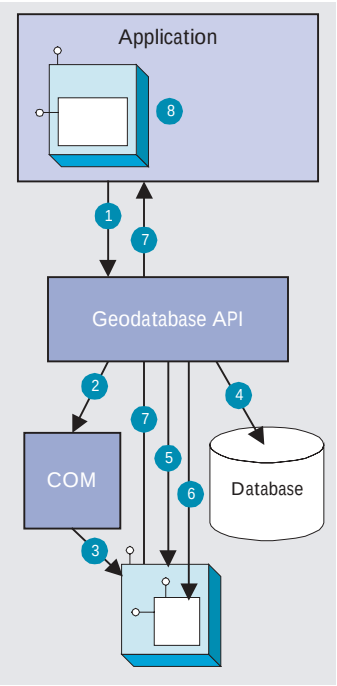
After changing an object, mark the object as changed (and ensure that it is updated in the database) by calling *Store* on the object. Delete an object by calling the *Delete* method on the object. Set versions of these calls also exist and should be used if the operation is being performed on a set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

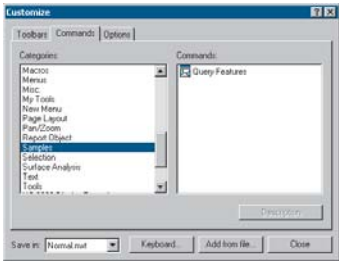
Note the extended data types at the bottom of the table: *VARIANT*, *BSTR*, and *VARIANT_BOOL*. Although it is possible to pass strings using data types such as *char* and *wchar_t*, these are not supported in languages such as Visual Basic. Visual Basic uses *BSTR*s as its text data type. A *BSTR* is a length-prefixed wide character array in which 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

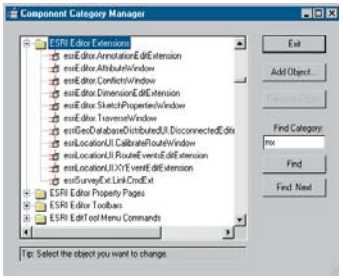
Component categories are used extensively in ArcObjects so developers can extend the system without requiring any changes to the ArcObjects code that will work with the new functionality.

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 the *IExtension* interface and is 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 debugging in Visual Basic is covered in 'The Visual Basic 6 development environment' section of this chapter, you'll see the importance of this.



The Customize dialog box in ArcMap and ArcCatalog



The Component Category Manager

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 provides 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 on the left), it is possible to choose 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. Although 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 on the left. This small application allows you to add and remove classes from any of the component categories on your system, not just ArcObjects categories. 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 of interacting 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 allows this functionality. See the sections on Visual Basic developer add-ins and Active Template Library (ATL) in the ArcGIS Developer Help.

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"). In addition, 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 control.

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 or a tab width of two for indentation. Since there is always only 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' in this chapter).

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

Recovery statements issued by error handlers

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 'The Microsoft Component Object Model' earlier in this chapter 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 accustomed 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 appears that 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 (VBE) 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 following code illustrates these concepts; note the reference count on the object at the various stages of code execution.

See the Visual Basic Magic sample on the disk for this code. You are encouraged to run the sample and use 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.*

```
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

  ' QI for a useful interface.
  ' Define the interface.
  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 let VB tidy up.
End Sub 'refCount = 0
```

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.

```
MapControl1.Map.Layers(0).Name = "Foo"
```

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

```
MapControl1.ActiveView.Extent.Width = 32
```

The reason that this does not work is that the VBVM expands the interface chain 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 = MapControl1.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

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 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.

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

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.

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*.

Coclass is an abbreviation of component object class.

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

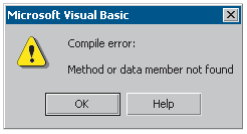
```
Set pPt = New esriGeometry.Point
```

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.*

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 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.



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 notifying the user, or it may mean automatically dealing with the error and continuing with the function. The choice depends on the circumstances. Below is a 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, and Boolean values. 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 be set either by reference or 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; *TrackCircle* returns an *IPolygon* interface.

```
Dim pCircle as IPolygon
set pCircle = MapControl1.TrackCircle
```

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 *IsActive* method of *IActiveView* returns a value of true if the map is active.

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 to which you can attach code 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, in which 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 = MapControl1.Map
```

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 as follows:

```
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 as 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. 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 the first layer in a map control, you can test whether it is a CAD layer using the following code:

```
  Dim pUnk As IUnknown
  Dim pCadLayer As ICadLayer

  Set pUnk = MapControl1.Layer(0)
```

```
If TypeOf pUnk Is ICadLayer Then
  Set pCadLayer = pUnk
  'do something with the layer
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. For an example, see the following code:

```
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. Imagine that you have 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.

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

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

```
Dim pEnv1 As IEnvelope
Dim pEnv2 As IEnvelope

Dim pActiveView1 As IActiveView
Dim pActiveView2 As IActiveView

Set pEnv1 = MapControl1.ActiveView.Extent
Set pEnv2 = MapControl1.ActiveView.Extent

Set pActiveView1 = MapControl1.ActiveView
Set pActiveView2 = MapControl1.ActiveView

'Extent returns a copy,
'so pEnv1 is pEnv2 returns false
MsgBox pEnv1 Is pEnv2

'ActiveView returns a reference so,
'ActiveView1 is ActiveView2 returns true
MsgBox pActiveView1 Is pActiveView2
```

Iterating through a collection

In your work with ArcObjects, 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 VB code that loops through the selected features (*IEnumFeature*) in a map control.

```
Dim pEnumFeat As IEnumFeature
Dim pFeat As IFeature
Set pEnumFeat = MapControl1.Map.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
```

The previous section of this chapter 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 Chapter 5, 'Licensing and deployment', in this guide.

This section is not intended as a Visual Basic tutorial; rather, it highlights aspects of Visual Basic that you should know 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 are 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 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.

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

5. Add any 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.

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.

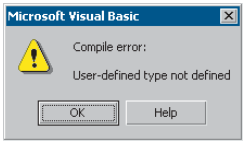
IMPLEMENTING INTERFACES

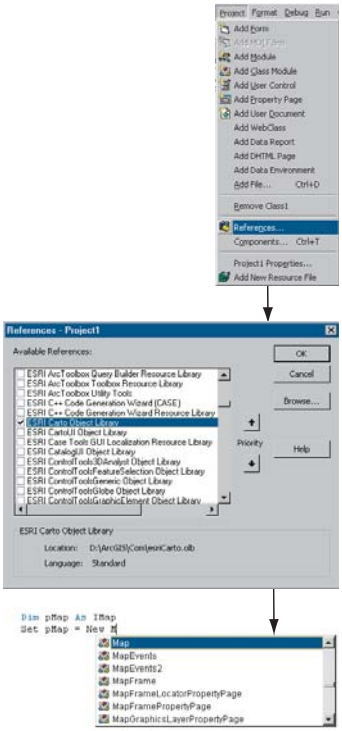
You implement interfaces differently in Visual Basic, depending on whether they are inbound or outbound interfaces. An outbound interface is seen by Visual Basic as an event source and is supported through the *WithEvents* keyword. To handle the outbound interface, *IActiveViewEvents*, in Visual Basic (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 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' in this chapter). To implement the *IExtension* interface, code similar to that 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
```





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

SETTING REFERENCES TO THE ESRI OBJECT LIBRARIES

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 shown on 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 in the developer help or by 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.

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 refer to the *Application*, 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 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 few objects support a reference to their parent object (the *IDocument* interface has a property named *Parent* that references the *IApplication* interface). 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 always run only 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("C:\Program Files\arcgis\bin\arcmap.exe _
  C:\Program Files\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.

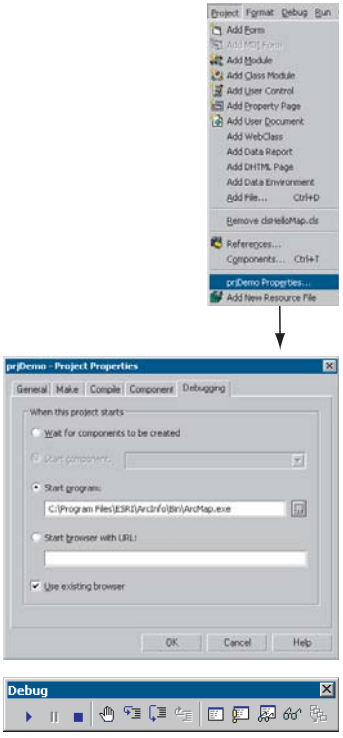
To execute a program and wait until it is terminated, you must call three Win32 API functions. First, call the *CreateProcessA* function to load and execute ArcMap. Next, call the *WaitForSingleObject* function, which forces the operating system to wait until ArcMap has been terminated. Finally, when 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 software-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.



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 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 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 marshalled. This marshalling causes problems in certain circumstances. Most data types can be automatically marshalled by the system, but there are a few that require custom code because the standard marshaller 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 marshalling 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, and it 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 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. Previously, it was discussed how component categories were used in this instance. Remember that 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, load ArcMap through the debugger.

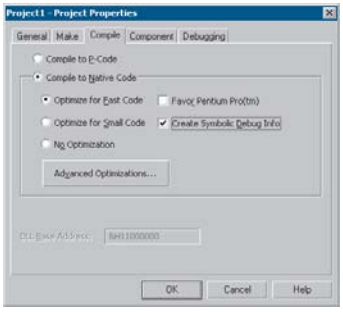
In some cases, such as extensions and property pages, it is not possible to use the Visual Basic debugger. If you have access to the Visual C++ debugger, you can use one of the options outlined below. Fortunately, there are a number of ESRI Visual Basic Add-ins that make it possible to track down the problem quickly and effectively. The add-ins, described in ArcGIS Developer Help 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
```

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 Visual C++ section 'Debugging tips in Developer Studio'. Both 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.



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

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 into the source file at the correct line. The required steps are as follows:

1. Start an appropriate application, such as ArcMap.exe.

2. Start Microsoft Visual C++.
3. Attach to the ArcMap process using the Menu options 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 wrapper 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 proceeds through 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, such as COM aggregation and singletons, that are possible in Visual C++ are not possible in Visual Basic 6. By using standard class libraries, such as 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 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 section 'ATL in Visual C++ .NET' later in this chapter.

With the addition of the Visual C# .NET language, it is worth considering porting Visual C++ 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:

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

WORKING WITH ATL

This section cannot cover all the topics that a developer working with ATL should know to be effective, but it will serve as an introduction to ATL. ATL helps you implement COM objects and 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 Microsoft Foundation Class Library (MFC) code, which provides more support for writing applications. An alternative to MFC is the Windows Template Library (WTL), which 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++ version 6 and Visual C++ .NET.

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, such as vectors, stacks, and queues. ATL also

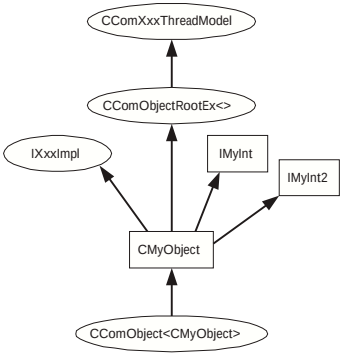
provides a set of wizards that extends the Visual Studio development environment. These wizards automate some of the tedious plumbing code that all ATL projects must have. The wizards include, but are not limited to, the following:

- Application—Used to initialize an ATL C++ project.
- Object—Used to create COM objects. Both C++ and IDL code are 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 including 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 to the left. The *CComxxxThreadModel* class supports thread-safe access to global, instance, and static data. The *CComObjectRootEx* class 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* interface contains 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 follows in the 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 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 the caller any indication of the nature of the failure. There are a number of Windows-standard HRESULTs available, for example, E_INVALIDARG (one or more arguments are invalid) and 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 following error will be generated during the build:

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

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.

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 9 runtime installation 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 Windows 9.x; 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, click Build > Configurations. Then delete *Win32 Debug*, *Win32 Release MinSize*, and *Win32 Release MinDependency*.

Registering a COM component

The ATL project wizard generates the standard Windows entry points for registration. This code will register the DLL's type library and execute a registry script file (.rgs) 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 the ATL's 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. The syntax describes keys, values, names, and subkeys to be added to 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 unregistration. *ForceRemove* will ensure the key and subkeys are removed before registering the new keys. The *s*, *d*, and *b* values indicate string (enclosed with apostrophes), double word (32-bit 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 that all COM objects use to register their ProgIDs 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, or END_CATEGORY_MAP. However, these macros do not correctly remove registry entries as explained in the Microsoft Developer Network (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 with 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 startup. 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.

```
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 nonzero represents true. However, the bool data type has now become part of ANSI C++. COM APIs are language independent and define a different Boolean type, VARIANT_BOOL. In addition, 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 optimize use. This type can also be promoted to an int value. Expressions (e.g., i!=0) return 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 a 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. 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. In addition, 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 an 8-bit ANSI character string (8-bit character) representation (found on Windows 95, Windows 98, and Windows Me platforms) to a 16-bit Unicode character string (16-bit 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, such as Windows NT, Windows 2000, and Windows XP, 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 (for example, SetWindowText) to resolve to an ANSI version (SetWindowTextA) or a Unicode version (SetWindowTextW). In addition, 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*, a 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 8-bit ANSI version that will interpret multibyte 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 16-bit characters plus a
                                     16-bit 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. Unlike C strings, they can contain embedded zero characters, although this is unusual. The BSTR also has a count value, which is stored four bytes before the BSTR pointer address. The CComBSTR wrappers are often used to manage the lifetime 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). In addition, 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 (such as an 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 is 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; otherwise, the stack may grow 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 noncreatable classes

Noncreatable 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 noncreatable class is returned. This type of object is found in abundance in the geodatabase model. For example, *FeatureClass* is noncreatable and can only be obtained by calling one of a number of methods; one example is the *IFeatureWorkspace::OpenFeatureClass* method.

One advantage of a noncreatable 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 noncreatable object:

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

// Bar is a noncreatable 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 noncreatable class are shown below:

1. Add "noncreatable" to the .idl file's 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 instances of the noncreatable class. This happens via ATL's 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)  // Noncreatable 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 noncreatable 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 noncreatable object Bar
  IBarPtr ipBar = 0;

  // C++ Pointer to Bar, with ATL template to supply CreateInstance
implementation
  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, there are enhancements and changes that are relevant to the ArcGIS ATL developer. Some of these are summarized 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 DLLs 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 boilerplate 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. Change this from Use Multi-Byte Character Set to Use Unicode Character Set.

Character conversion macros—The character conversion macros (USES_CONVERSION, W2A, W2CT, and so forth) 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. These macros 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 files atlstr.h and cstringt.h. If ATL is combined with MFC, this defaults to MFC's CString implementation.

Filepath handling—A collection of related functions for processing the components of filepaths is available through the CPath class defined in atlpath.h.

ATLServer—This is a new selection of ATL classes designed for writing Web applications, XML Web services, and other server applications.

#import issues—When using #import, a few modifications are required. 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

The Microsoft Developer Network 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*

You may also find the following books to be 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, Reilly Stockton, 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, Chris Sells, and Jim Springfield. *ATL Internals*. Reading, MA: Addison–Wesley, 1999.

SMART TYPES

Smart types are objects that behave as 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 are 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 smart types covered here are Direct-To-COM and 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 objects.
- *_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'.

It is possible to create an object implicitly in a DTC smart pointer's constructor, for example:

```
IFooPtr ipFoo(CLSID_Foo)
```

However, this will raise a C++ exception if there is an error during object creation—for example, if the DLL 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, 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 and CComVariant are typically used.

ATL smart types include:

- *CComPtr*—encapsulates a COM interface pointer by wrapping the *AddRef* and *Release* methods of the *IUnknown* interface
- *CComQIPtr*—encapsulates a COM interface and supports all three methods of the *IUnknown* interface: *QueryInterface*, *AddRef*, and *Release*
- *CComBSTR*—encapsulates the *BSTR* data type
- *CComVariant*—encapsulates the *VARIANT* data type
- *CRegKey*—provides methods for manipulating Windows registry entries
- *CComDispatchDriver*—provides methods for getting and setting properties and calling methods through an object's *IDispatch* interface
- *CSecurityDescriptor*—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();
```

The equality operator ("==") may have different implementations when used during smart pointer comparisons. The COM specification states object identification 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, so you must use the* ATL IsEqualObject *method.*

```
  if (FAILED(hr)) return hr;

  // Explicitly call Release().
  ipGak = 0;
  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. *CComVariant* derives directly from the VARIANT data type, meaning that there is no overloading with its implementation, which in turn simplifies its 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 Visual C++, 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 so it 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:

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.

```
STDMETHODIMP CFoo::get_Name(BSTR* name)
{
  if (name==0) return E_POINTER;
  CComBSTR bsName(L"FooBar");
  *name = bsName.Detach();
}
```

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, that is, types that are not themselves smart types with overloaded cast operators.

```
// Perform QI of IUnknown.
IUnknownPtr ipUnk = ipSmartPointer;
// Ensure IUnknown* constructor of CComVariant is used.
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 caller's 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 shown 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 zero pointer is required before AddRef can be called. Calling AddRef (or any method) on a zero 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 nonzero before calling AddRef.
  if (*pVal)
    *pVal->AddRef();

  return S_OK;
}
```

When using a smart pointer to receive an object from an [out] parameter on a

method, use the smart pointer "&" dereference 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);

 // Initialize ipBar Smart pointer from Foo1.
 IBarPtr ipBar;
 ipFoo1->get_Bar(&ipBar);

 // The "&" dereference will call Release on ipBar.
 // ipBar is then repopulated 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 a separate line:

```
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 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 the statement to bring up the debugging context menu). Then 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 the Settings command of the project menu. Click the C/C++ tab, then click General in the Category dropdown list. In the Debug info dropdown list, click Program Database for Edit and Continue.

Unicode string display

To 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 the Eyeglasses icon or press Shift+F9 to bring up the QuickWatch dialog box 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 by right-clicking it and toggling 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—for instance, when a for loop reaches 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, then click the Condition button to display a dialog box in which you specify the break point condition.

Preloading DLLs

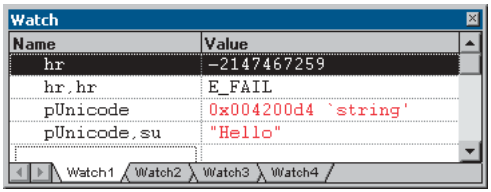
You can preload DLLs that you want to debug before executing the program. This allows you to set break points up front rather than waiting until the DLL has been loaded during program execution. To do this, click Project, click Settings, click Debug, click Category, then click Additional DLLs. Then, click in the list area to add any DLLs you want to preload.

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 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.



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.)O&.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 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 to which a register points, 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. Useful 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

To reference ArcGIS interfaces, types, and objects, you will need to import the definitions into Visual C++ 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 library types, 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 and GUID constants (LIBID, CLSID, and IID) and to define smart pointers. The exclude (OLE_COLOR, OLE_HANDLE, VARTYPE) is required because Windows 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 type library header (.tlh) files, which are similar in format to a .h file. You may also find a type library implementation (.tli) file, which corresponds to a .cpp file. These files can be large but are only regenerated when the type libraries change.

There are many type libraries at ArcGIS 9 for different functional areas. You can start by 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 of type libraries required:

1. Do a compilation and look at the "missing type definition" errors generated from code, for example, 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, such as WKSPoint or interfaces that are inherited into other interfaces. For example, if working with geometry objects, such as points, start by importing esriGeometry. The compiler will issue various errors such as:

   ```
   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 "C:\Program Files\ArcGIS\com\esriSystem.olb" raw_interfaces_only,
raw_native_types, no_namespace, named_guids, exclude("OLE_COLOR",
"OLE_HANDLE", "VARTYPE")
  #import "C:\Program Files\ArcGIS\com\esriSystemUI.olb"
raw_interfaces_only, raw_native_types, no_namespace, named_guids
  #import "C:\Program Files\ArcGIS\com\esriGeometry.olb"
raw_interfaces_only, raw_native_types, no_namespace, named_guids
  #import "C:\Program Files\ArcGIS\com\esriDisplay.olb" raw_interfaces_only,
raw_native_types, no_namespace, named_guids
  #import "C:\Program Files\ArcGIS\com\esriOutput.olb" raw_interfaces_only,
raw_native_types, no_namespace, named_guids
  #import "C:\Program Files\ArcGIS\com\esriGeoDatabase.olb"
raw_interfaces_only, raw_native_types, no_namespace, named_guids
  #import "C:\Program Files\ArcGIS\com\esriCarto.olb" raw_interfaces_only,
raw_native_types, no_namespace, named_guids

  // Some of the Engine controls
  #import "C:\Program Files\ArcGIS\bin\TOCControl.ocx" raw_interfaces_only,
raw_native_types, no_namespace, named_guids
  #import "C:\Program Files\ArcGIS\bin\ToolbarControl.ocx"
raw_interfaces_only, raw_native_types, no_namespace, named_guids
  #import "C:\Program Files\ArcGIS\bin\MapControl.ocx" raw_interfaces_only,
raw_native_types, no_namespace, named_guids
  #import "C:\Program Files\ArcGIS\bin\PageLayoutControl.ocx"
raw_interfaces_only, raw_native_types, no_namespace, named_guids

  // Additionally for 3D controls
  #import "C:\Program Files\ArcGIS\com\esri3DAnalyst.olb"
raw_interfaces_only, raw_native_types, no_namespace, named_guids
  #import "C:\Program Files\ArcGIS\com\esriGlobeCore.olb"
raw_interfaces_only, raw_native_types, no_namespace, named_guids
  #import "C:\Program Files\ArcGIS\bin\SceneControl.ocx"
raw_interfaces_only, raw_native_types, no_namespace, named_guids
  #import "C:\Program Files\ArcGIS\bin\GlobeControl.ocx"
raw_interfaces_only, raw_native_types, no_namespace, named_guids

  #pragma warning (pop)
```

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("C:\Program Files\ArcGIS\com\esriSystemUI.olb");

 coclass ZoomIn
  {
   [default] interface IUnknown;
```

```
    interface ICommand;
  }
};
```

For a general discussion of ATL, see the earlier section 'ATL in brief'.

ATL AND THE ACTIVEX CONTROLS

This section covers how to use ATL to add controls to a dialog box. Although ATL is focused on providing COM support, it also supplies some useful Windows programming wrapper classes. One of the most useful is CWindow, 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
```

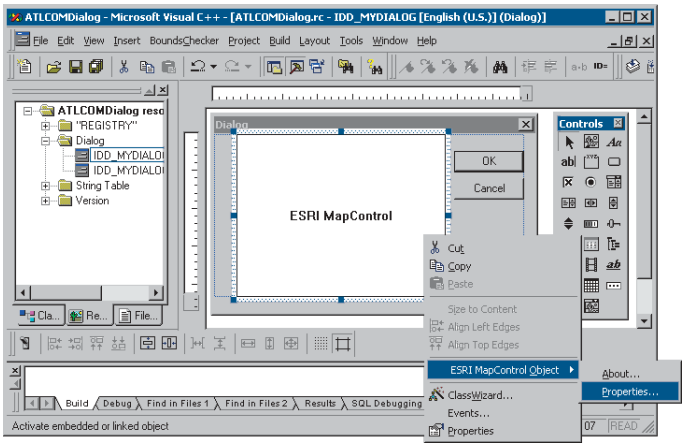
CWindow is a generic wrapper for all window handles, so for specific Windows messages to window common controls, such as buttons, tree views, or edit boxes, one approach is to send window messages directly to the window, for example:

```
// Set button to be checked (pushed in or checkmarked, 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 article "HOWTO: Using Class Wrappers to Access Windows Common Controls in ATL", available for download from Microsoft. This header file is an early version of Windows Template Libraries.

Visual Studio Resource Editor can be used to design and position Windows common controls and ActiveX controls on a dialog box. To create and manipulate the dialog box, 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 boilerplate code. The steps to create a dialog box and add an ActiveX control in an ATL project are discussed below.

1. Click the menu command Insert/New ATL Object.
2. Click the Miscellaneous category, then click the Dialog object.
3. A dialog box resource and a class inheriting from CAxDialogImpl will be added to your project.
4. Right-click the dialog box in Resource view and click Insert ActiveX Control. This will display a list of available ActiveX controls.
5. Double-click a control in the list to add that control to the dialog box.
6. Right-click the control and click Properties to set the control's design-time properties.



Make sure dialog boxes that host ActiveX controls inherit from CAxDialogImpl *and not* CDialogImpl. *If this mistake is made, the DoModal method of the dialog box simply exits with no obvious cause.*

Make sure applications that use windows common controls, such as treeview, correctly call InitCommonControlsEx *to load the window class. Otherwise, the class will not function correctly.*

Accessing a control on a dialog box through a COM interface

To retrieve a handle to the control that is hosted on a form, use the *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 initializes 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 to use the class wizard. Right-click the control and choose Events. Next, click the resource ID of the control, then click the event (for example, *OnMouseDown*). Next click Add Handler. Finally, ensure the dialog box begins listening to events by adding AtlAdviseSinkMap(this,TRUE) to *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 later 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 box. This is typically performed in the *OnInitDialog* windows message handler of a dialog box.

```
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

Disabled buddy property on property page

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 the *OnInitDialog* method.

ToolbarControl not resized 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 disappearing when displaying context-sensitive help

When viewing the controls property page at design time, right-clicking and clicking "What's This?" will cause 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 for how to work with ArcGIS ActiveX controls in Visual C++, the first of which is what framework to use to host the controls (for example, ATL or MFC). A second decision is where the control will be hosted (Dialog, MDI application, and so forth). This section discusses MFC and hosting the control on a dialog box.

Creating an MFC dialog box-based application

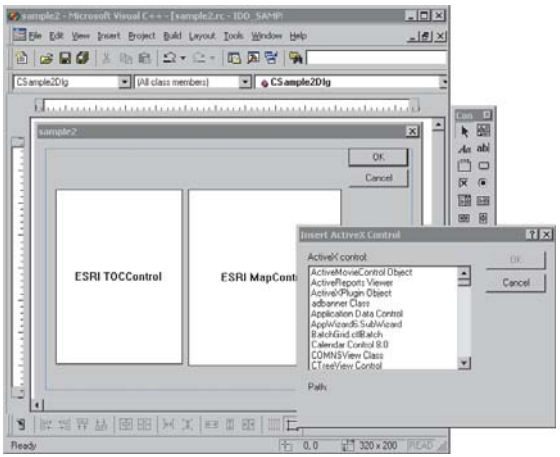
If you do not have a dialog box in your application or component, here are the steps to create an MFC dialog box application.

1. Launch Visual Studio C++ 6 and click New.
2. Click the Projects tab and choose MFC AppWizard (exe). Enter the project name and location and click OK.
3. For Step 1 of the wizard: From the radio buttons, change the application type to Dialog Based. Click Next.
4. For Step 2 of the wizard: The default project features are fine, although you can uncheck AboutBox to simplify the application. Ensure that the option to support ActiveX controls is checked. Click Next.
5. For Step 3 of the wizard: The default settings on this page are fine. The MFC DLL is shared. Click Next.
6. For Step 4 of the wizard: This shows you what the wizard will generate. Click Finish.

You should now have a simple dialog box-based application. In the resource view, you will see "TODO: Place Dialog Controls Here". You can place buttons, list boxes, and so forth, in this dialog box, and the dialog box can also host 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 box and accessing them using IDispatch

1. Right-click the MFC dialog box and click Insert ActiveX control.
2. Double-click a control from the list box. The control appears on the dialog box with a default size.



Inserting ActiveX controls on a dialog box in Visual Studio C++ Design time. The TOCControl *and* MapControl *have been added to the dialog box. The* ToolbarControl *is next.*

3. Size and position the control as required.
4. Repeat steps 1 through 3 for each control.
5. You can right-click the control and choose Properties to set the control's design-time properties.
6. To access the control in code, you will need ArcGIS interface definitions for *IMapControl*, for example. To do this use the #import command in your stdafx.h file. See the section 'Importing ArcGIS type libraries' on how to do this.
7. MFC provides control hosting on a dialog box; this will translate Windows messages, such as WM_SIZE, into appropriate control method calls. However, to be able to make calls on a control, there are a few steps you must perform to go from a resource ID to a controls interface. The following code illustrates setting the *TOCControl*'s Buddy to be the *MapControl*:

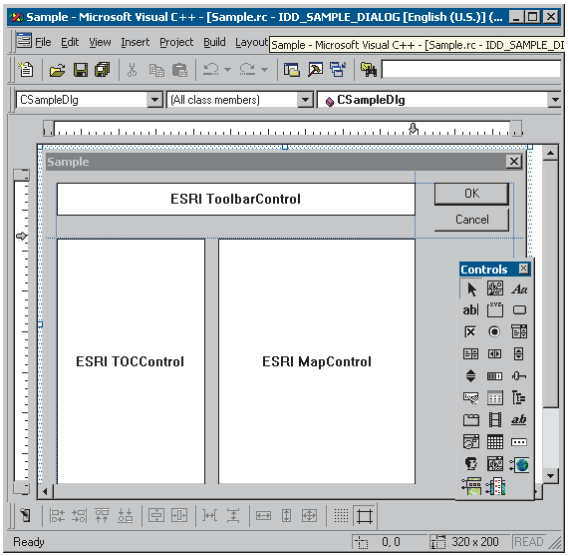
```
// Code to set the Buddy property of the TOCControl to be the MapControl
// Get a pointer to the PageLayoutControl and TOCControl.
IPageLayoutControlPtr ipPageLayoutControl;
GetDlgControl(IDC_PAGELAYOUTCONTROL1, IID_IPageLayoutControl, (void**)
            &ipPageLayoutControl);
ITOCControlPtr ipTOCControl;
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, double-click 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's name from becoming "OnOnMouseDownMapcontrol1". The wizard will then automatically generate the necessary MFC sink map macros to listen to events.

Adding controls to an MFC dialog box using IDispatch wrappers

As all ActiveX controls support IDispatch, this is the typical approach to add an ActiveX control to an MFC project:

1. Click Project, click Add, then click Components and Controls.
2. Click Registered ActiveX Controls.



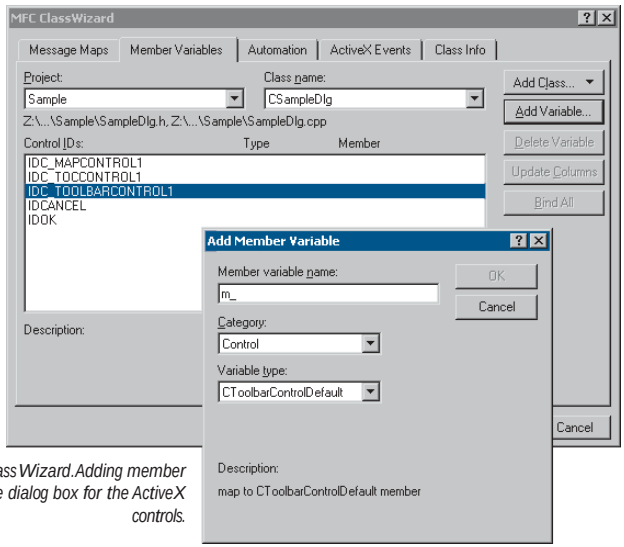
The design environment showing the TOCControl, MapControl, *and* ToolbarControl *has been added to the Controls toolbar and to the dialog box.*

3. Double-click to select a control (for example, ESRI TOCControl), then click OK to insert a component. Click OK to generate wrappers. This will add a button for the control to the Controls toolbar in Visual Studio.
4. Additional source files are added to your project (for example, toccontrol.cpp and toccontrol.h). These files contain a wrapper class (for example, *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 an MFC *CWnd* class that hosts an ActiveX control.
5. Repeat steps 1 through 4 to add each control to the project's Controls toolbar.
6. Choose a control from the Controls toolbar and drag it onto the dialog box.
7. Right-click the control and click Properties. 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 using the *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 the control and choose Class Wizard to launch the class wizard. Click the Member Variables tab and click the resource ID corresponding to the control to give the control a member variable name. The dialog box class member variable can now be used to invoke methods and properties on the control.



Visual Studio C++ ClassWizard.Adding member variables to the dialog box for the ActiveX controls.

9. To catch control events, click the Message Maps tab of the class wizard and choose the resource ID of the control. In the list of messages, click the event to catch—for example, *OnBeginLabelEdit*. Double-click this event and a handler for it will be added to your dialog box 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*.

Do not use the method GetIDispatch *(inherited from MFC's* CCmdTarget*) on the wrapper classes; it is intended for objects implementing* IDispatch *and not the wrapper classes that are calling* IDispatch*. Instead, to get a control's* IDispatch*, use* m_mapcontrol.GetControlUnknown() *followed by* QueryInterface *to* IDispatch*. See the above example of setting the Buddy property.*

HANDLING COM EVENTS IN ATL

Below 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 that 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 *Map* coclass 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 because 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 that 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 the standard COM interfaces *IConnectionPointContainer* and *IConnectionPoint*.

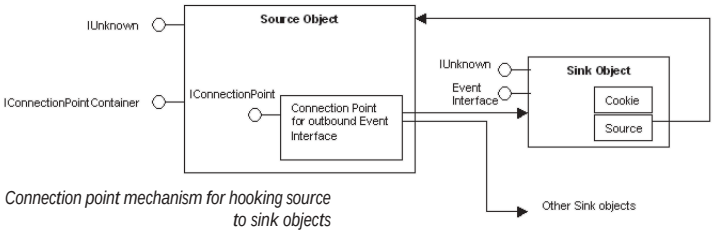
Fire event—When a source object needs to inform all the sinks of a particular action, the source is said to fire an event. This results in the source iterating all the sinks and making the same method call on each. For example, when a layer is added to a map, the *Map* coclass is said to fire the *ItemAdded* 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 the *IConnectionPointContainer* interface to allow 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 QI the source object's *IConnectionPointContainer* and call *FindConnectionPoint* to supply an interface ID for outbound interfaces. 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* to represent 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, the sink will be given an identifier—a simple unsigned long value called a cookie—to give back to the source at a later point when it no longer needs to listen to events.

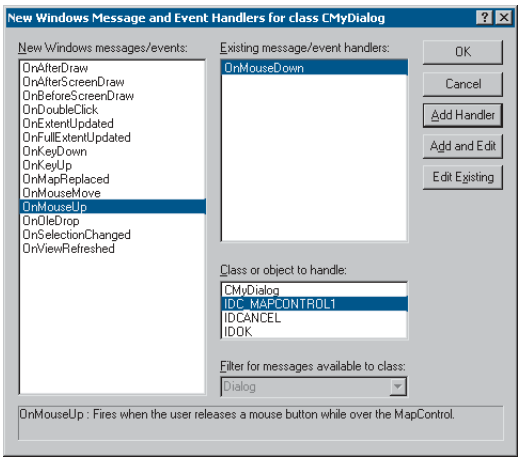


Connection point mechanism for hooking source to sink objects

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. Also, Microsoft Visual Basic 6 can only be a source of pure *IDispatch* events. The connection point mechanism is the same as for pure COM mechanisms, the main difference being in how the events are fired.



Visual Studio C++ ClassWizard.Adding event handler to an ActiveX control on a dialog box.

ATL provides some macros to assist with listening to *IDispatch* events; this is discussed on MSDN under 'Event Handling and ATL'. There are two templates available, *IDispEventImpl* and *IDispEventSimpleImpl*, that 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 control to a dialog box. Right-click the Control and click Events. In the Class wizard, choose the resource ID of the control, choose the event, then click Add Handler.

The following code 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: it does not add the advise and unadvise code to the dialog box. 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 the 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!"));
  }
```

The following issues with events are documented on the 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,* 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)

```
BEGIN_SINK_MAP(CMyDialog)
  // Make sure the Event Handlers have __stdcall calling convention.
  // 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 now requires the developer to supply a pointer to a structure describing the format of the event parameters. This structure is typically placed in the .cpp file. For example, here is the structure describing the parameters of an *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
};
```

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
```

See the 'Importing ArcGIS type libraries' section earlier in this chapter for an explanation of #import.

```
        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 box, 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 implementation of CListen contains the following code to start listening to the controls; the typedef 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.
```

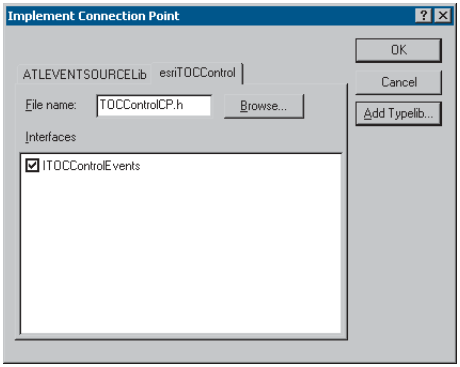
The implementation of CListen also contains the following code to UnAdvise and stop listening to the controls.

```
// 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

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. In addition, 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 ensure that 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 the COM object and click Implement Connection Point.
3. Either use a definition of events from the IDL in the project or click Add Typelib to browse for another definition.
4. Check the outbound interface to be implemented in the coclass.



5. Clicking OK will modify your 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, use the following example, which illustrates a coclass that is a source of *ITOCControlEvents*, 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 in 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, and 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; this must be checked 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, call *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
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*. In addition, a coclass can have a default outbound interface; this is specified with the *[default]* tag. Default interfaces are identified by some design environments (for example, Visual Basic 6). Following 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 to fire events, and 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 released and may cause substantial memory leaks. There are a number of ways to tackle this issue:

- Ensure the advise and unadvise are made on a method or Windows message that is guaranteed to happen in pairs and is independent of an object's life cycle. For example, in a coclass that is also receiving Windows messages, use the Windows messages *OnCreate* (WM_CREATE) and *OnDestroy* (WM_DESTROY) to advise and unadvise.
- If an ATL dialog box class needs to listen to events, one approach is to make the dialog box a private COM class and implement the events interface directly on the dialog box. ATL allows this without much extra coding. This approach is illustrated below. The dialog box class creates a *CustomizeDialog* coclass and listens to *ICustomizeDialogEvents*. The *OnInitDialog* and *OnDestroy* methods (corresponding to Windows messages) are used to advise and unadvise on *CustomizeDialog*.

```
class CEngineControlsDlg :
  public CAxDialogImpl<CEngineControlsDlg>,
  public CComObjectRoot, // Make Dialog Class a COM Object as well.
  public ICustomizeDialogEvents  // Implement this interface directly on
                                    this object.

CEngineControlsDlg() : m_dwCustDlgCookie(0) {} // Initialize 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 box.
  STDMETHOD(OnStartDialog)();
  STDMETHOD(OnCloseDialog)();

  ICustomizeDialogPtr     m_ipCustomizeDialog;  // The source of events
  DWORD                 m_dwCustDlgCookie;    // Cookie for
                                                  CustomizeDialogEvents.
}
```

The dialog box needs to be created like a noncreatable 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 box alive until you're done with it.
myDlg->DoModal();   // Launch the dialog box; when method returns, dialog
                       box has exited.
myDlg->Release();   // Typically, the refcount now goes to 0 and frees the
                       dialog object.
```

- 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 events to the sink object. The listener has its reference count incremented by the source, but the sink's 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 destructor code, it instructs 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 that is a weak reference. That is, the listener contains a COM pointer to the sink but does not increment the sink's reference count. It is the responsibility of the sink to ensure that this pointer is not accessed after the sink object has been released.

This section,'What is the .NET Framework?', summarizes the Microsoft overview of the .NET Framework available online as part of the MSDN Library.The complete text is available at http://www.msdn.microsoft.com.

WHAT IS THE .NET FRAMEWORK?

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:

- Provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- Provide a code execution environment that minimizes software deployment and versioning conflicts.
- Provide a code execution environment that guarantees safe execution of code, including code created by an unknown or semitrusted third party.
- Provide a code execution environment that eliminates the performance problems of scripted or interpreted environments.
- Make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- 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 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 semitrusted 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.

Regarding security, managed components are awarded varying degrees of trust, depending on a number of factors that includes 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 onscreen 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 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 Windows 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 and reporting tools. 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 or C++ in conjunction with the Microsoft Foundation Classes 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 re-creates 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 semitrusted 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

This section does not address licensing considerations and is intended only to illustrate the possibilities of server application development using the .NET API. While the ArcGIS Engine Developer Kit can be used to develop applications that run on single-use computers and that may or may not leverage ArcSDE-, ArcGIS Server-, or ArcIMS-based services, custom components deployed on a server require an ArcGIS Server license. Contact your ESRI regional office or international distributor for more information.

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.

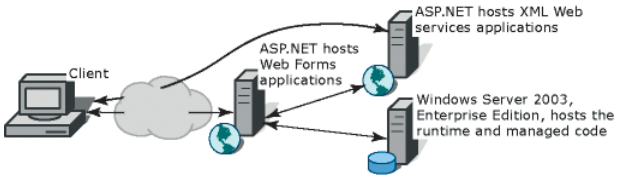
Server-side managed code

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 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 Active Server Pages (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.



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 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 .NET Framework interacts with COM via a technology known as COM Interop.

For COM Interop to work, the Common Language Runtime (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

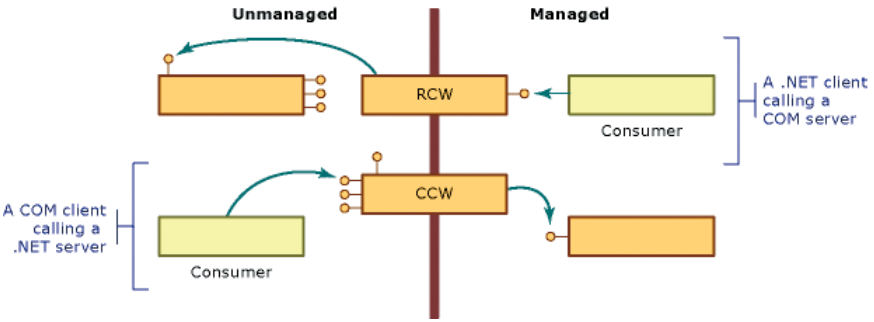
Primary interop assemblies (PIAs) are the official, vendor-supplied, .NET type definitions for interoperating with underlying COM types. Primary interop assemblies are strongly named by the COM library publisher to guarantee uniqueness.

ESRI provides primary interop assemblies for all the ArcObjects type libraries that are implemented with COM. ArcGIS .NET developers should only use those primary interop assemblies that 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).

The ArcGIS installation program also installs the Microsoft Stdole.dll PIA, providing interop for OLE font and picture classes, which are used by some ESRI libraries.

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) that handles the marshalling between the two environments. Similarly, the .NET runtime creates COM-callable wrappers for the reverse case, COM clients communicating with .NET components. The illustration above outlines this process.



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 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 small and often unnoticeable. However, the cost of creating wrappers and having them marshall 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 that managed code can invoke. This improves performance by limiting the marshalling 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 that has the following naming convention: ESRI.ArcGIS plus the name of the 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 also 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 event sinks.

The .NET classes also have class members that .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 implements 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 adding the interface name and an underscore as a prefix 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, then the word EventHandler. For example, the *SelectionChanged* event defined on the *IActiveViewEvents* interface has the following delegate defined: *IActiveViewEvents_SelectionChangedEventHandler*. The importer also creates an event interface with an '_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 noncompliant methods, and these cannot be used in .NET. However, in most cases, supplemental interfaces have been added that 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);
```

Class Interfaces

Class interfaces are created to help VB programmers transition to .NET; they are also commonly used in code produced by the Visual Basic .NET Upgrade Wizard or the code snippet converter in Visual Studio .NET.

However, it is generally recommended that you avoid using the class interfaces in the ESRI interop assemblies, as they may change in future versions of ArcGIS. This section explains a little more about class interfaces.

In Visual Basic 6, the details of default interfaces were hidden from the user, and a programmer could instantiate a variable and access the members of its default interface without performing a specific *QI* for that interface; for example, the VB 6 code below instantiates the *StdFont* class and sets a variable equal to the default interface (*IStdFont*) of that class:

```
[VB 6.0]
Dim fnt As New Stdole.StdFont
```

However, .NET does not provide this same ability. To allow VB developers a more seamless introduction to .NET, the type library importer in .NET adds 'class interfaces' to each interop assembly, allowing COM objects to be used with this same syntax inside .NET. When an object library is imported, a class interface RCW is created for each COM class; the name of the class interface is the same as the COM class—for example, Envelope.

All the members of the default interface of the COM class are added to this class interface; also, if the COM class has a source interface (is the source of events), then the class interface will also include all the events of this interface, which helps a programmer to link up events.

A second RCW is created that represents the underlying COM class; the name of this is the same as the COM class with a suffix of 'Class', for example, *EnvelopeClass*. The class interface is linked to the class by an attribute, which indicates the class to which it belongs. This attribute is recognized by the .NET compilers, which allows a programmer to instantiate a class by using its class interface.

The exception is classes that have a default interface of *IUnknown* or *IDispatch*, which are never exposed on RCW classes as the members are called internally by the .NET Framework runtime. In this case, the next implemented interface is exposed on the class interface instead. As most ArcObjects define *IUnknown* as their default interface, this affects most ArcObjects classes. For example, the Point COM class in the esriGeometry object library lists the *IPoint* interface as its first implemented interface. In .NET, this class is accessed by using the *Point* class interface, which inherits the *IPoint* interface, and the *PointClass* class.

The code below shows that by declaring a variable type as a *Point* class interface,

that variable can be used to access the *IPoint::PutCoords* method from this class interface.

```
[VB.NET]
Dim thePt As ESRI.ArcGIS.Geometry.Point = New ESRI.ArcGIS.Geometry.Point()
thePt.PutCoords(10,8)
[C#]
ESRI.ArcGIS.Geometry.Point thePt = newESRI.ArcGIS.Geometry.Point();
thePt.PutCoords(10,8);
```

The inherited interface of a class interface is not guaranteed to remain the same between versions of ArcGIS and, therefore, it is recommended that you avoid using the above syntax.

You can view these types in the VB .NET Object Browser. Notice that using Visual Basic .NET, *PointClass* is not shown by default but can be made visible by selecting the Show Hidden Members option. In the C# Object Browser, you can see more clearly the class interface *Point* and its inherited interface *IPoint* and the class *PointClass*.

.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 no additional syntax, whereas explicit casts require cast operators.

```
geometry = point                    ' Implicit cast
geometry = CType(point, IGeometry)  ' Explicit cast
```

When casting between interfaces, it's perfectly acceptable to use implicit casts because 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 recommended 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 to use is 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 the 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 *GUIDAttribute* 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. The event handler 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 command wiring up to a MapControl 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_map As Map
  Private m_pHookHelper As IHookHelper

  Public Overloads Overrides Sub OnCreate(ByVal hook As Object)
  m_pHookHelper.Hook = hook

  End Sub

  Public Overrides Sub OnClick()
  m_map = m_pHookHelper.FocusMap
```

```
    ' Create an instance of the delegate, and add it to SelectionChanged event.
    SelectionChanged = New
IActiveViewEvents_SelectionChangedEventHandler(AddressOf OnSelectionChanged)
    AddHandler m_map_SelectionChanged, SelectionChanged

  End Sub

  ' Event handler
  Private Sub OnSelectionChanged()
    MessageBox.Show("Selection Changed")
  End Sub
```

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 that inherits from the System.Exception base class. Many different types of exception classes 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 of 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 of code.
}
```

You place a try block around code that 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 that must always be executed, for example, to

clean up resources such as 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, the exact outcome may depend on the location of the executed code and the configuration of the .NET runtime. Therefore, it is advisable to include at least 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 want; 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 VB6. 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.

Therefore, it is advisable 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 want to attempt to correct the error condition in code and try the call again. Alternatively, you may want to report the error to the user to let them decide which course of action to take; 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 want to deal with an error by creating a specific error condition and propagating this error to the caller. You can do this using the Throw keyword.

To throw the existing error to the caller function, write your error handler using the Throw keyword, as shown below.

```
[VB.NET]
Catch ex As System.Exception
...

[C#]
catch (System.Exception ex)
{
  throw;
}
...
```

If you want 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. 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.

System.__ComObject and casting to strongly typed RCWs

Sometimes you may find that casting a variable fails when you think it should succeed (the solution is often to declare variables as interface Types and avoid the use of class types, for example, use *IStyleGallery* rather than *StyleGalleryClass*). You may also have come across the System.__ComObject type and wonder where it comes from. This topic should help you to understand these issues; in particular, you may encounter problems when attempting to create the AppRef class in .NET—this issue is related to the System.__ComObject wrapper and is also covered below.

Types and Runtime Callable Wrappers

In .NET, each class, interface, enumeration, and so on, is described by its *Type*. The Type class, which is part of the .NET Framework, holds information about

the data and function members of a datatype. When you create a new COM object in .NET via interop, you get a reference to your object that is wrapped in a strongly typed runtime callable wrapper (RCW). A RCW is a wrapper that can hold a reference to a COM object inside a .NET application.

To understand what this means, look at the following code extract; a variable called sym is declared as the *ISimpleMarkerSymbol* interface Type and is then set to a new *SimpleMarkerSymbolClass*. Then the Type of the variable sym is retrieved and written to the debug window. If you were to run this code, you would find that the Type of sym is *SimpleMarkerSymbolClass*, as you might expect; the variable holds a reference to the *ISimpleMarkerSymbol* interface of the *SimpleMarkerSymbolClass* RCW.

```
[C#]
ESRI.ArcGIS.Display.ISimpleMarkerSymbol sym = new
ESRI.ArcGIS.Display.SimpleMarkerSymbolClass();
Debug.WriteLine(sym.GetType().FullName);
[Visual Basic .NET]
Dim sym As ESRI.ArcGIS.Display.ISimpleMarkerSymbol = New
ESRI.ArcGIS.Display.SimpleMarkerSymbolClass
Debug.WriteLine(CType(sym, Object).GetType.FullName)
```

In a different coding situation, you may get a reference to a RCW from another property or method. For example, in the similar code below, the Symbol property of a renderer (*ISimpleRenderer* interface) is retrieved, where the renderer uses a single *SimpleMarkerSymbol* to draw.

```
[C#]
ESRI.ArcGIS.Display.ISimpleMarkerSymbol sym = rend.Symbol as
ESRI.ArcGIS.Display.ISimpleMarkerSymbol;
Debug.WriteLine(sym.GetType().FullName);
[Visual Basic .NET]
Dim sym As ESRI.ArcGIS.Display.ISimpleMarkerSymbol = rend.Symbol
Debug.WriteLine(CType(sym, Object).GetType.FullName)
```

Although you might expect to get the same output as before, you will actually find that the reported Type of sym is System.__ComObject.

The System.__ComObject Type

The difference between the two excerpts of code above is that in the first you create the symbol using the New (or new) keyword and the Type SimpleMarkerSymbolClass. When the code is compiled, the exact Type of the variable is discovered by the compiler using Reflection, and *metadata* about that Type is stored in the compiled code. When the code runs, the runtime then has all the information (the metadata) that describes the exact Type of the variable.

However, in the second example, you set the sym variable from the Symbol property of the *ISimpleRenderer* interface. When this code is compiled, the only metadata that the compiler can find is that the Symbol property returns an *ISymbol* reference; the Type of the actual class of object cannot be discovered. Although you can perform a cast to get the *ISimpleMarkerSymbol* interface of the sym variable (or any other interface that the symbol implements), the .NET runtime does not have the metadata required at runtime to discover exactly what the Type of the variable is. In this case, when you access the Symbol property, the

.NET runtime wraps the COM object reference in a generic RCW called System.__ComObject. This is a class internal to the .NET Framework that can be used to hold a reference to any kind of COM object; its purpose is to act as the RCW for an unknown Type of COM object.

Casting

Looking again at the second example, even if you actually know the exact Type of class to which you have a reference, the .NET runtime still does not have the metadata required to cast the variable to a strongly typed RCW; this can be seen in the following code, as attempting a cast to the *SimpleMarkerSymbolClass* Type would fail.

```
[C#]
// The following line would result in sym2 being null as the cast would
fail.
ESRI.ArcGIS.Display.SimpleMarkerSymbolClass sym2 = sym as
ESRI.ArcGIS.Display.SimpleMarkerSymbolClass;
[Visual Basic .NET]
' The following line would result in a runtime error as the implicit cast
would fail.
Dim sym2 As ESRI.ArcGIS.Display.SimpleMarkerSymbol = sym
```

However, as the System.__ComObject class is specifically designed to work with COM objects, it is always able to perform a QI to any COM interfaces that are implemented by an object. Therefore, casting to specific interfaces (as long as they are implemented on the object) will be successful.

```
[C#]
ESRI.ArcGIS.Display.ISimpleMarkerSymbol sym3 = sym as
ESRI.ArcGIS.Display.ISimpleMarkerSymbol;
[Visual Basic .NET]
Dim sym3 As ESRI.ArcGIS.Display.ISimpleMarkerSymbol = sym
```

Singletons and System.__ComObject

In the examples above, a strongly typed RCW is created when you instantiate the COM object by using the 'new' keyword, whereas if the object is preexisting, the Type of the RCW is the generic System.__ComObject. Sometimes when you use the 'new' keyword to instantiate a COM object, you are actually getting a reference to an object that already exists—this happens when you attempt to instantiate a singleton class that has previously been instantiated. The .NET framework is unable to wrap in a strongly typed RCW an instance of an object that has previously been wrapped in the generic System.__ComObject RCW. If your code has encountered such a situation, you may receive an error such as

'Unable to cast object of type System.__ComObject to type <Typename>'.

```
[C#]
ESRI.ArcGIS.Display.IStyleGallery sg = new
ESRI.ArcGIS.Framework.StyleGalleryClass();
[Visual Basic .NET]
Dim sg As ESRI.ArcGIS.Display.IStyleGallery = = New
ESRI.ArcGIS.Framework.StyleGalleryClass
```

This error may occur even though you have declared your variable using the interface name rather than the class name, as shown above. The problem occurs

because when your code instantiates an object, the .NET runtime first attempts to wrap the object in the strongly typed class Type (the Type stated after the new keyword) before attempting a cast to the interface type. The cast to the strongly typed RCW cannot succeed as the COM object has previously been wrapped in the generic System.__ComObject wrapper. This may occur in situations beyond your control. For example, other ArcObjects tools written in .NET from other third parties may wrap an object in the generic wrapper, causing your code to fail.

The solution is to use the Activator class (as shown below) to safely wrap singletons objects in a strongly typed RCW when you first get a reference to them. Additionally, you should generally always declare variables holding RCWs using an interface rather than a class Type.

Using the Activator class to create singletons

If you use the CreateInstance method of the Activator class instead of the new keyword to instantiate singletons, you can avoid such errors, as the Activator is able to get the required metadata to perform the cast.

```
[C#]
Type t = Type.GetTypeFromProgID("esriFramework.StyleGallery");
System.Object obj = Activator.CreateInstance(t);
IStyleGallery sg = obj as IStyleGallery;
[Visual Basic .NET]
Dim t As Type = Type.GetTypeFromProgID("esriFramework.StyleGallery")
Dim obj As System.Object = Activator.CreateInstance(t)
Dim pApp As ESRI.ArcGIS.Display.IStyleGallery = obj
```

You can use this technique to instantiate the AppRef class—remember, however, that the AppRef class can only be created within an ArcGIS application. (The Type is the generic System.__ComObject RCW.)

```
[C#]
Type t = Type.GetTypeFromProgID("esriFramework.AppRef");
System.Object obj = Activator.CreateInstance(t);
ESRI.ArcGIS.Framework.IApplication pApp = obj as
ESRI.ArcGIS.Framework.IApplication;
[Visual Basic .NET]
Dim t As Type = Type.GetTypeFromProgID("esriFramework.AppRef")
Dim obj As System.Object = Activator.CreateInstance(t)
Dim pApp As ESRI.ArcGIS.Framework.IApplication = obj
```

For more information about RCWs and interop, you may wish to refer to the book by Adam Nathan, *.NET and COM—The Complete Interoperability Guide*, Sams Publishing, 2002.

Working with resources

Using strings and embedded images directly (no localization)

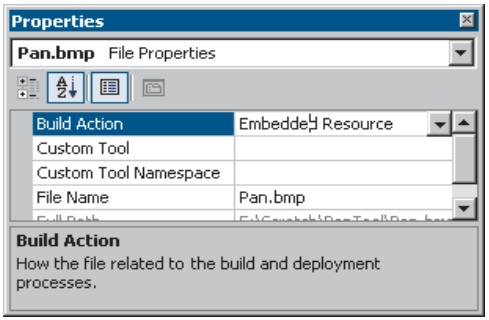
If your customization does not support localization now and you do not intend for it to support localization later, you can use strings and images directly without the need for resource files. For example, strings can be specified and used directly in your code:

```
[VB.NET]
```

```
Me.TextBox1.Text = "My String"

[C#]
this.textBox1.Text = "My String";
```

Image files (BMPs, JPEGs, PNGs, and so forth) can be embedded in your assembly as follows:

1. Right-click the project in the Solution Explorer, click Add, then click Add Existing Item.
2. In the Add Existing Item dialog box, browse to your image file and click Open.
3. In the Solution Explorer, select the image file you just added, then press F4 to display its properties.
4. Set the Build Action property to Embedded Resource.



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 resource files

Before attempting to provide localized resources, you should ensure you are familiar with the process of creating resource files for your .NET projects. Even if you do not intend to localize your resources, you can still use resource 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, and so forth) as long as 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 into either the main project assembly or a separate satellite assembly that contains only resources.

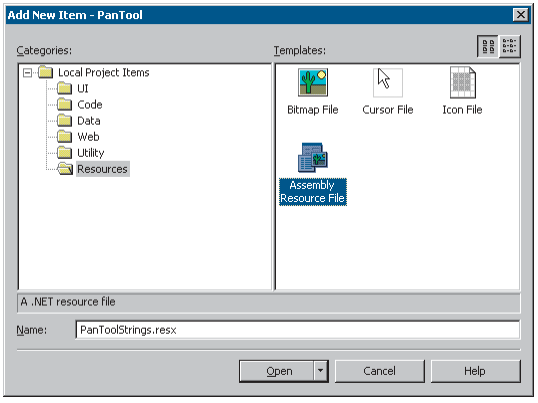
The following options are available to create your resource files. Each is discussed below.

- Creating a .resx file for string resources
- Creating resource files for image resources
- Compiling a .resx file into a .resources file

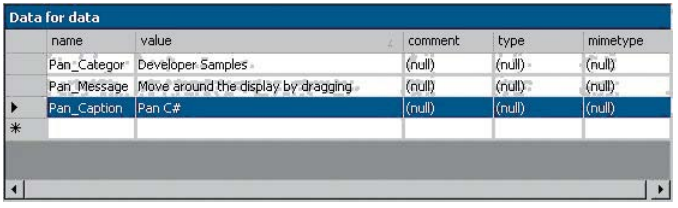
Creating a .resx file for string resources

If all you need to localize is strings—not images or cursors—you can use Visual Studio .NET to create a new .resx file that will be compiled automatically into a .resources module embedded in the main assembly.

1. Right-click the project name in the Solution Explorer, click Add, then click Add New Item.
2. In the Add New Item dialog box, click Assembly Resource File.



3. Open the new .resx file in Visual Studio, and add name–value pairs for the culture-specific strings in your application.



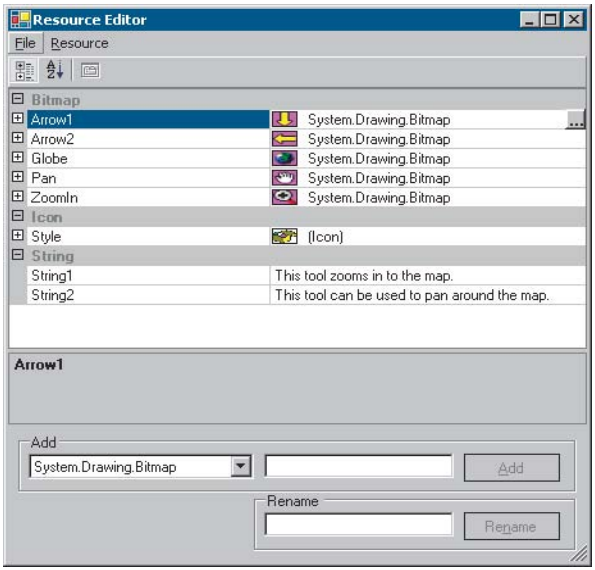
4. When you compile your project, the .resx file will be compiled into a .resources module inside your main assembly.

Creating resource 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.

However, a number of sample projects are available with the Visual Studio .NET Framework SDK that can help you work with resource files. One such sample is the Resource Editor (ResEditor).

The ResEditor sample can be used to add images, icons, imagelists, and strings to a resource file. The tool cannot be used to add cursor resources. Files can be saved as either .resx or .resource files.



Creating resource 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 want to add resources that cannot be handled by the .NET Framework SDK samples and tools, for example, cursors. The basic usage of the two classes is similar: first, create a new resource writer class specifying the filename, 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 cursor to the file.

[VB.NET]

A list of tools useful for working with resources can be found in the Microsoft .NET Framework documentation.

```
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();
```

Additional information on the ResEditor sample can be found in the Microsoft .NET Framework documentation.

The PanTool developer sample (Samples\Map Analysis\Tools) includes a script—MakeResources—that shows you how to use the *ResXResourceWriter* class to write bitmaps, cursor files, and strings into a .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.

The ResEditor sample is provided by Microsoft as source code.You must build the sample first if you want to create resource files using this tool. You can find information on building the SDK samples under the SDK subdirectory of your Visual Studio .NET installation.

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 the ResX Generator (ResXGen) sample in the tutorial.

- 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.
- 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 want 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.

More information on the ResXGen can be found in the Microsoft .NET Framework documentation.

When you build a .NET project that 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 subdirectories of the main build directory. The subdirectories are named after the culture of the satellite assembly they contain. For example, Swiss–French resources would be contained in a fr-CH subdirectory.

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.

Embedding a default .resources file in your project

1. Right-click the project name in the Solution Explorer, click Add, then click Add Existing Item to navigate to your .resx or .resources file.
2. In the Solution Explorer, choose the file you just added and press 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, for example, <BaseName>.<Culture>.resx or <BaseName>.<Culture>.resources.
3. Add the new resource 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 subdirectories under the directory holding your main assembly. The subdirectories will be named by culture, allowing the .NET runtime to locate the resources appropriate to the culture in which the application runs.

   The main (default) resources file will be embedded in the main assembly.

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.

Assembly versioning and redirection

When new ArcGIS libraries are installed onto a machine, the corresponding primary interop assemblies (PIA) are updated as well to ensure that the available PIAs always correspond to the current libraries.

.NET applications that are built using a specific version of a strongly named assembly will attempt to bind to the same version of that assembly at runtime because a strong name includes the version number of the assembly.

To allow applications that bind to ArcGIS assemblies to find the new PIAs without the need for developers to recompile against these new versions, ArcGIS installs publisher policy files for each PIA. A publisher policy file is an assembly that redirects the assembly binding process to use a new assembly version. These policy files are installed to the general assembly cache (GAC) alongside the ESRI PIAs.

For example, you will find the policy file policy.9.0.ESRI.ArcGIS.System.dll, which redirects all assembly binding calls for the ESRI.ArcGIS.System.dll PIA from all 9.0 and service pack assembly versions to the new current assembly version.

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, a standalone application that uses the MapControl, or a custom layer. This section discusses several key issues related to developing with ArcGIS and .NET.

Deploying an application using XCOPY will not copy the settings in the machine configuration file.

Registering .NET components with COM

Extending 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 Visual Studio .NET, which has a Register for COM Interop compiler flag.

The example below shows an EditTools assembly being registered with COM. The /tlb parameter specifies that a type library should also 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, click Project Properties from the Project menu, then look at the Build property under Configuration Properties. The last item, Register for COM Interop, should be set to True.

Registering .NET classes in COM component categories

Much of the extensibility of ArcGIS relies on COM component categories. In fact, most custom ArcGIS components must be registered in component categories appropriate to their intended context and function 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 box

Custom .NET ArcGIS commands and tools can quickly be added to toolbars via the Add From File button on the Customize dialog box. In this case, you simply have to browse for the TLB and open it. The ArcGIS framework will automatically add 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, browse for your type library, and choose the appropriate class.

COM Register Function

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 that 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 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 is greatly simplified. For more details on using these classes, see the 'Working with the ESRI .NET component category classes' section below.

Simplifying 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 also 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 set the members of the base class directly.

The base classes expose their internal member variables to the inheritor class, one per property, so 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, as *OnClick* will not do anything by default. This may be suitable for a tool but is probably not for a command.

To override any member, you can right-click the member of the base class in the Solution Explorer window, click Add, then click Override to stub out the member as overridden. Note that if you right-click 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, click Overrides from the dropdown list on the right in the Code Window Wizard bar, then choose the member you want to override from the left dropdown 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 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 that

MxCommands.Register and MxCommands.Unregister are used 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);
  }
```

Working with OLE StdFont and StdPicture classes

Some ArcObjects libraries make use of classes and interfaces defined within the standard OLE libraries from Microsoft. To use these members within .NET, you should add to your project a reference to the Stdole.dll primary interop assembly, which is included as part of the .NET support during an ArcGIS installation. This PIA allows you to define StdFont and StdPicture classes, for example:

```
[C#]
stdole.IFontDisp fnt = (stdole.IFontDisp) new stdole.StdFontClass();
fnt.Name = "Arial";
fnt.Size = 20.0F;
ESRI.ArcGIS.Display.TextSymbol textSym = new
  ESRI.ArcGIS.Display.TextSymbolClass();
textSym.Font = fnt;

[Visual Basic .NET]
Dim fnt As stdole.IFontDisp = New stdole.StdFontClass()
fnt.Name = "Arial"
```

```
fnt.Size = 20.0
Dim textSym As ESRI.ArcGIS.Display.TextSymbol = New
  ESRI.ArcGIS.Display.TextSymbolClass()
textSym.Font = fnt
```

Sometimes, however, you may have an existing .NET Font, Bitmap, or Icon class that you wish to convert to use as a Font or Picture in an ESRI method. The ESRI.ArcGIS.Utility.COMSupport namespace, which is part of the ESRI.ArcGIS.Utility.dll assembly, provides the OLE class, which may help you to perform such conversions.

Note that these members depend on the System.Windows.Forms.AxHost class and as such are only suitable for use within a project that has a reference to the System.Windows.Forms.dll assembly.

Below you can find the syntax information for the members of the ESRI.ArcGIS.Utility.COMSupport.OLE class—these are static (shared in VB.NET) members and, therefore, can be called without the need to instantiate the OLE class.

GetIFontDispFromFont: This method can be used to convert an existing .NET System.Drawing.Font object into an Stdole.StdFont object.

```
[C#]
  public static object GetIFontDispFromFont (System.Drawing.Font font)

[Visual Basic .NET]
  Public Shared GetIFontDispFromFont (ByVal font As System.Drawing.Font) As
Object
```

GetIPictureDispFromBitmap: This method can be used to convert an existing .NET System.Drawing.Bitmap object into an Stdole.StdPicture object.

```
[C#]
  public static object GetIPictureDispFromBitmap (System.Drawing.Bitmap bitmap)

[Visual Basic .NET]
  Public Shared GetIPictureDispFromBitmap (ByVal bitmap As
    System.Drawing.Bitmap) As Object
```

GetIPictureDispFromIcon: This method can be used to convert an existing .NET System.Drawing.Icon object into an Stdole.StdPicture object.

```
[C#]
  public static object GetIPictureDispFromIcon (System.Drawing.Icon icon)

[Visual Basic .NET]
  Public Shared GetIPictureDispFromIcon (ByVal icon As System.Drawing.Icon)
As Object
```

Below are some examples of using the members of the OLE class.

```
[C#]
System.Drawing.Font dotNetFont = new System.Drawing.Font("Castellar", 25.0F);
ESRI.ArcGIS.Display.ITextSymbol textSym = new
ESRI.ArcGIS.Display.TextSymbolClass() as ESRI.ArcGIS.Display.ITextSymbol;
textSym.Font =
ESRI.ArcGIS.Utility.COMSupport.OLE.GetIFontDispFromFont(dotNetFont) as
stdole.IFontDisp;
```

```
System.Drawing.Bitmap dotNetBmp = new
System.Drawing.Bitmap(@"C:\Temp\MyBitmap.bmp");
ESRI.ArcGIS.Display.IPictureMarkerSymbol bmpSym = new
ESRI.ArcGIS.Display.PictureMarkerSymbolClass() as
ESRI.ArcGIS.Display.IPictureMarkerSymbol;
bmpSym.Picture =
ESRI.ArcGIS.Utility.COMSupport.OLE.GetIPictureDispFromBitmap(dotNetBmp) as
stdole.IPictureDisp;

System.Drawing.Icon dotNetIcon = new
System.Drawing.Icon(@"C:\Temp\MyIcon.ico");
ESRI.ArcGIS.MapControl.IMapControlDefault  map = this.axMapControl1.Object
as ESRI.ArcGIS.MapControl.IMapControlDefault;
map.MouseIcon =
ESRI.ArcGIS.Utility.COMSupport.OLE.GetIPictureDispFromIcon(dotNetIcon) as
stdole.IPictureDisp;
map.MousePointer =
ESRI.ArcGIS.SystemUI.esriControlsMousePointer.esriPointerCustom;

[Visual Basic .NET]
Dim dotNetFont As New System.Drawing.Font("Castellar", 25.0F)
Dim textSym As ESRI.ArcGIS.Display.ITextSymbol = New
ESRI.ArcGIS.Display.TextSymbolClass
textSym.Font =
ESRI.ArcGIS.Utility.COMSupport.OLE.GetIFontDispFromFont(dotNetFont)

Dim dotNetBmp As System.Drawing.Bitmap = New
System.Drawing.Bitmap("C:\Temp\MyBitmap.bmp")
Dim bmpSym As ESRI.ArcGIS.Display.IPictureMarkerSymbol = New
ESRI.ArcGIS.Display.PictureMarkerSymbolClass
bmpSym.Picture =
ESRI.ArcGIS.Utility.COMSupport.OLE.GetIPictureDispFromBitmap(dotNetBmp)

Dim dotNetIcon As System.Drawing.Icon = New
System.Drawing.Icon("C:\Temp\MyIcon.ico")
Dim map As ESRI.ArcGIS.MapControl.IMapControlDefault =
Me.AxMapControl1.Object
map.MouseIcon =
ESRI.ArcGIS.Utility.COMSupport.OLE.GetIPictureDispFromIcon(dotNetIcon)
map.MousePointer =
ESRI.ArcGIS.SystemUI.esriControlsMousePointer.esriPointerCustom
```

Shutting down ArcGIS .NET applications

To help unload COM references in .NET applications, the *AOUninitialize* class provides the static (shared in VB.NET) function Shutdown. This class is part of the ESRI.ArcGIS.Utility.COMSupport namespace in the ESRI.ArcGIS.Utility.dll assembly.

For more information on shutting down ArcGIS .NET applications, see 'Releasing COM References' in this chapter.

```
[C#]
ESRI.ArcGIS.Utility.COMSupport.AOUninitialize.Shutdown();

[Visual Basic .NET]
```

```
ESRI.ArcGIS.Utility.COMSupport.AOUninitialize.Shutdown()
```

This section does not address licensing considerations and is intended only to illustrate the possibilities of server application development using the .NET API. While the ArcGIS Engine Developer Kit can be used to develop applications that run on single-use computers and that may or may not leverage ArcSDE-, ArcGIS Server-, or ArcIMS-based services, custom components deployed on a server require an ArcGIS Server license. Contact your ESRI regional office or international distributor for more information.

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 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 marshalled using the Automation marshaller. You specify this by adding *AutomationProxyAttribute* to your class with a value of true.
- Your COM class should generate a dual class interface. You specify this by adding *ClassInterfaceAttribute* 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 shut down. 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 avoid crashes, 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 *AOUninitialize.Shutdown* function handles most of the shutdown problems in standalone applications, but you may still experience problems as there are COM objects that require explicit releasing; in these cases, call

System.Runtime.InteropServices.Marshal.ReleaseComObject to decrement the reference count, allowing 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]
 Sub Main()
  Dim styCls As ESRI.ArcGIS.Display.IStyleGallery = New
    ESRI.ArcGIS.Framework.StyleGalleryClass
  ' Use the StyleGalleryClass here ...
  Release(styCls)
 End Sub

 Sub Release(ByVal comObj As Object)
  Dim refsLeft As Integer = 0
   Do
    refsLeft = System.Runtime.InteropServices.Marshal.ReleaseComObject(comObj)
  Loop While (refsLeft > 0)
 End Sub
[C#]
 private void MyFunction()
 {
  ESRI.ArcGIS.Display.IStyleGallery styCls = new
    ESRI.ArcGIS.Framework.StyleGalleryClass() as
    ESRI.ArcGIS.Display.IStyleGallery;
  // Use the StyleGalleryClass here ...
  Release(styCls as object);
 }

 void Release(object comObj)
 {
  int refsLeft = 0;
   do
   {
    refsLeft = Marshal.ReleaseComObject(comObj);
   }
  while (refsLeft > 0);
 }
```

Working with geodatabase cursors in ArcGIS Server

This section does not address licensing considerations and is intended only to illustrate the possibilities of server application development using the .NET API. While the ArcGIS Engine Developer Kit can be used to develop applications that run on single-use computers and that may or may not leverage ArcSDE-, ArcGIS Server-, or ArcIMS-based services, custom components deployed on a server require an ArcGIS Server license. Contact your ESRI regional office or international distributor for more information.

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 on to 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 on to an SDE stream, and if the application has multiple clients, each may get and hold on to 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 that services 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 SubSystem.object 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

Note that .NET Support is a separate option in the ArcGIS installation; this needs to be selected during installation on both the development and target machines for .NET customizations to succeed. If you did not install .NET Support originally, you can run the installation program again and choose the Modify option to add features to your ArcGIS installation.

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 or setup program. Note that aside from the ArcGIS primary interop assemblies and the .NET Framework assemblies, all dependencies must also be packaged and deployed.

ArcGIS components

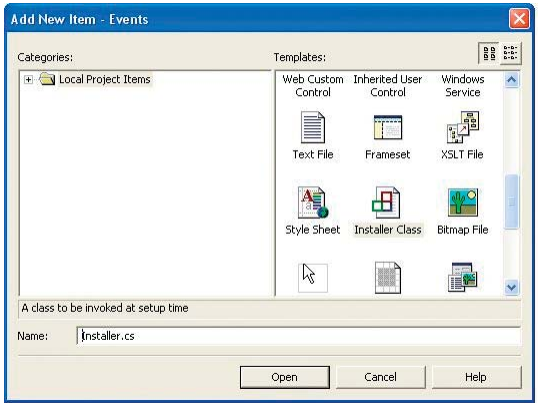
Components that extend the ArcGIS applications are trickier to deploy than standalone applications because they must be registered with COM and in specific component categories. As discussed earlier, implementing *COMRegisterFunction* and *COMUnregisterFunctions* facilitates deployment by providing self 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 or 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.

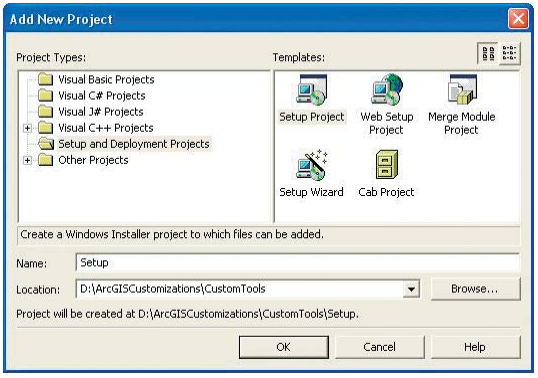


Override the Install and Uninstall functions that are implemented in the Installer base class and use the *RegistrationServices* class's *RegisterAssembly* and *UnregisterAssembly* methods to register the components. Make sure you use the SetCodeBase flag; this indicates that the code base key for the 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.



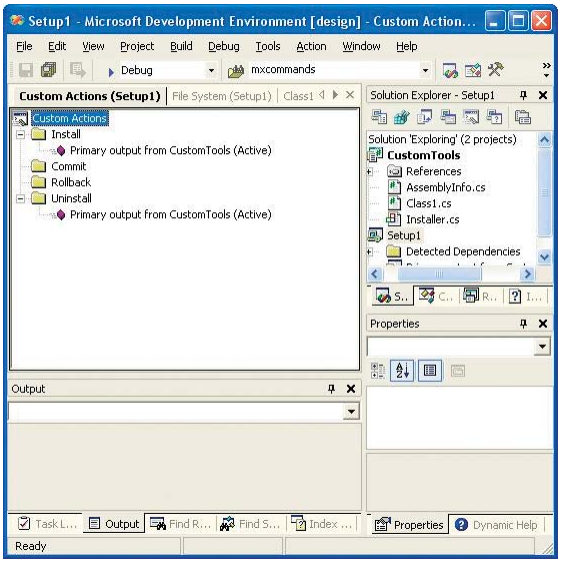
   a. In the Solution Explorer, right-click the new project and click Add > Project Output. Choose the project you want to deploy and choose Primary output.

   b. From the list of detected dependencies that is regenerated, remove all references to ESRI primary interop assemblies (for example, ESRI.ArcGIS.System) and stdole.dll. The only items typically left in the list are your TLB and Primary output from <AssemblyName><Version>', which represent 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 the setup project in the Solution Explorer and click View Custom Actions.

d. In the resulting view, right-click the Install folder and click Add Custom Action. Double-click the Application folder, then double-click the Primary output from the <AssemblyName><Version> item. This step 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 file. 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 your project.
2. Click the Project menu, then click Copy Project.
3. In the Copy Project dialog box, 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 (SOC) machines.

The ArcGIS API for the Java platform is a programming interface that 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 that 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—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 specifications provided for 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 class path and environment settings.

Java developer kit

To develop with ArcObjects using the ArcGIS API for Java, 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
   ```

2. Edit the PATH variable to include the bin directory of JAVA_HOME.

   ```
   PATH=...;%JAVA_HOME%\bin
   ```

The PATH environment variable is a list of directory paths for executables such as javac, java, and javadoc. When an executable is specified without a path, this variable is used to help locate that executable.

To compile server-based applications, such as servlets and EJBs, you will also need to install and include the Java 2 Enterprise Edition toolkit in your class path. Java application servers generally provide this, or you can get the reference implementation provided by Sun Microsystems.

ArcGIS Engine

ArcGIS API for Java developer kit uses standard Java Native Interface (JNI) to access core ArcObjects components. This requires some native libraries to be in your path when compiling and running applications. You must be sure to include the correct paths to invoke interoperability into native ArcObjects. The native *.dll or *.so files are located in the bin subdirectory under ArcGIS.

On Windows, your path should include ...\ArcGIS\bin.

ESRI recommends setting an ARCENGINEHOME environment variable. Although this is not a requirement to use the ArcGIS Engine Developer Kit, the developer samples all use this variable to ensure your class path settings are accurate.

On Solaris and Linux, ARCENGINEHOME and all required ArcGIS paths are set by sourcing one of the init_engine scripts.

- When using C-shell, use this command:

```
source /path/to/arcgis/init_engine.csh
```

- When using bash or bourne shell, use this command instead:

```
source /path/to/arcgis/init_engine.sh
```

Setting the ARCENGINEHOME variable on Windows:

```
ARCENGINEHOME=[path to ArcGIS install directory]
```

For example:

```
ARCENGINEHOME=c:\ArcGIS
```

Editing the path 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.

On Windows:

```
PATH=...;%ARCENGINEHOME%\java\jre\bin
```

Class path

The ArcGIS API for Java provides Java Archive (JAR) files and the native runtime libraries. These JAR files are located on disk at %ARCENGINEHOME%\java on Windows and $ARCENGINEHOME/java on Solaris and Linux.

All Java applications built with any of the ArcGIS developer kits must have the jintegra.jar file referenced in the respective application's class path. This file contains all of the class files for the runtime library that handle interoperability to COM.

In addition, individual arcgis_xxx.jar files should be added to your class path as needed. For example, applications that leverage the PageLayout bean included with ArcObjects require that the arcgis_pagelayoutbean.jar file be added to the class path.

As an optional convenience, you can use the JARs in %ARCENGINEHOME%\java\opt or $ARCENGINEHOME/java/opt instead of a long list of arcgis_xxx.jar files. These two convenience JARs are:

- arcobjects.jar—contains all of the non-UI class files in one complete archive.
- arcgis_visualbeans.jar—contains all of the UI class files in one complete archive.

ANT

The ArcGIS SDK for Java includes numerous sample applications, each of which is delivered with ANT scripts. ANT is a Java-based build tool that uses build

scripts written in XML. You can either load these samples into your preferred development environment or use the ANT scripts to compile and run them.

If you want to use ANT scripts, you should have a working ANT installation. For the ArcGIS Engine samples, any ANT version 1.5.3 or higher will work. For your convenience, a version of ant, called arcgisant, is included under the ArcGIS\DeveloperKit\tools directory on Windows and the arcgis/developerkit/tools directory on Solaris and Linux.

The ANT scripts included with the samples require certain additional environment settings:

- ANT_HOME should point to your ANT installation directory. For example:

```
ANT_HOME=C:\ArcGIS\DeveloperKit\tools\ant
```

Solaris and Linux developers can source arcgis/developerkit/tools/setenv_ant.csh or setenv_ant.sh, which sets this variable and includes arcgisant's bin directory in the PATH.

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 *.dll files in the bin directory and the necessary *.jar files 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 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 takes 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 interacts with the operating system. The JVM handles such things 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 ArcGIS API for Java.

To see how the initializeEngine *method is used as the first call, refer to the Java developer samples in ArcGIS Developer Help.*

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*.

```
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 ArcGIS API for Java 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 ArcGIS API for Java provides proxy classes that are generated from ArcObjects components type libraries, which allow interoperability with the underlying components. These proxy classes expose ArcObjects properties, methods, and events via their Java equivalents.

Import directives

Java import statements allow fully qualified class names to be shortened to their simple names. The code snippets in the following sections use simple class names and assume the corresponding import statements are in effect:

```
import com.esri.arcgis.system.beans.reader.*;
import com.esri.arcgis.system.datasourcesfile.*;
import com.esri.arcgis.system.*;
import com.esri.arcgis.system.geodatabase.*;
import com.esri.arcgis.system.geometry.*;
```

Multiplatform development

For multiplatform compatibility, the data and pathnames you use must be in all lowercase letters. You will encounter problems if any letters are uppercase.

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. 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 ArcGIS API for Java 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:

ArcObjects Interface	Java Representation
interface IArea : IUnknown	public interface IArea{} public class IAreaProxy implements IArea{}

The proxy classes are used internally by the ArcGIS API for Java 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 ArcGIS API for Java has two ways of accessing objects—by interface or by class.

```
/* use the class implementing com.esri.arcgis.geometry.Point();
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.

ArcObjects Interface	Java Representation
interface IGeometry : IUnknown	public interface IGeometry{}
interface IGeometry2 : IGeometry	public interface IGeometry2 extends IGeometry{}
interface IGeometry3 : IGeometry2	public interface IGeometry3 extends IGeometry2{}
interface IGeometry4 : IGeometry3	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 class types 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 ArcGIS API for Java. 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 type can be created as a property of another class or instantiated by objects from another class. In the ArcGIS API for Java, the default constructor normally used to create a class is undefined for ArcObjects classes.

```
/* The constructor for FeatureClass() is unsupported. */
FeatureClass fc = new FeatureClass(); // incorrect usage
```

The following example illustrates this behavior while guiding 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 through 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.

ArcObjects Struct	Java Representation

struct WKSPointZ
  double x
  double y
  double z
}
	public class _WKSPointZ {
	  public double x;
	  public double y;
	  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

Versions of the Java 2 SDK prior to version 5 do 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.

ArcObjects Struct	Java Representation

enum esri3DAxis
esriXAxis = 0
esriYAxis = 1
esriZAxis = 2
}
	public interface esri3DAxis {
	public static final int esriXAxis = 0;
	public static final int esriYAxis = 1;
	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 ArcGIS API for Java.

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 on 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 values. 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 same 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 in your code 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, if you use the same code with the ArcGIS API for Java, 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 ArcGIS API for Java has a constructor that takes in a single object as a 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 ArcGIS API for Java, 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 *poly* 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 ArcGIS API for Java, 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 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(10, 10);

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.Java*. When you construct a *Point* object, a proxy class for each implemented interface is also constructed. This allows you to use *instanceof* on any of these types. Developers would have access to any methods on *Point* 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 superinterfaces.

Consider the *IWorkspaceFactory.openFromFile* method, which returns an *IWorkspace*. Since the object returned is a Java object that 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 that 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 ArcGIS API for Java requires sending single element arrays as parameters 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,
    IARControl arControl,
    JLabel statusLabel) throws IOException {
  /*
  * Create two single-dimension arrays of type double to serve as
  * "out" parameters in a call to toMapPoint.
  */
  double[] dXcoord = {0.0};
  double[] dYcoord = {0.0};
  int screenX = params.getX();
  int screenY = params.getY();
  IARMap arMap = arControl.getARPageLayout().getFocusARMap();
  arMap.toMapPoint(screenX, screenY, dXcoord, dYcoord);
  statusLabel.setText("Map x,y: " + dXcoord[0] + ", " + dYcoord[0]);
}
```

The ArcGIS API for Java 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 superinterface to *IPolyline*, and the *Polyline* class implements both interfaces. In the first attempt you tried to send a single element *Polyline* array into a method that 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 superclass 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

A few ArcObjects types are not OLE automation-compliant. They contain methods that do not work in the ArcGIS API for Java. The API addresses each of these situations in one of two ways:

1. Supplemental interfaces have been added that have the offending methods overwritten in an automation-compliant way. These new interfaces are named by appending them with the letters "GEN", implying that they are generic for all supported APIs. In these cases, the noncompliant interface is deprecated with a link to the appropriate GEN interface. In the following example, "somePoints" is an array of Point with two Point objects in it.

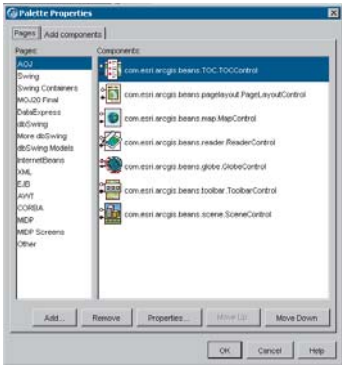
   ```
   /* Not automation compatible - throws exception */
   IEnvelope env = new Envelope();
   env.defineFromPoints(2, somePoints[0]);

   /* Automation compatible */
   IEnvelopeGEN envGEN = new Envelope();
   envGEN.defineFromPoints(somePoints);
   ```

2. A singleton utility class implements bridge interfaces. Since each utility class can handle methods from multiple noncompliant interfaces, there is no naming convention for the utility classes. The bridge interfaces handle the effort of converting between noncompliant and compliant methods. In these cases, the noncompliant methods are deprecated, with a link to the appropriate bridge method in the utility class. To call the bridge method, simply pass in the noncompliant object as the first argument. In this example, myPointCollection4 is an instance of IPointCollection4, and somePointStructures is an array of WKSPointZ objects with four objects in it.

   ```
   /* Not automation compatible - throws exception */
   myPointCollection4.addWKSPointZs (4, somePointStructures[0]);

   /* Automation compatible */
   GeometryEnvironment geomEnv = new GeometryEnvironment();
   geomEnv.addWKSPointZs(myPointCollection4, somePointStructures);
   ```

Using visual beans

The ArcGIS API for Java 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 En-



gine application. The use of beans creates a bridge between Java and the ActiveX controls provided by ArcGIS. These visual components are Java Swing components that contain heavyweight AWT components. They conform to the JavaBeans component architecture, allowing them to be used as drag-and-drop components for designing Java GUIs in JavaBean-compatible IDEs. To successfully use the ArcGIS Java Beans, the static initializer *EngineInitializer.initializeVisualBeans* must be the first method call in *Main*. Due to the initialization dependencies of the ArcGIS Engine SDK for Java, your application is not guaranteed to function properly if usage of *EngineInitializer.initializeVisualBeans* is not correct.

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 JavaBeans for ArcObjects 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 JavaBeans for ArcObjects 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 uses 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 document*/
    java.awt.EventQueue.invokeLater(
      new Runnable() {
        public void run() {
          statusLabel.setText(
          " Document filename: "+ arControl.getDocumentFilename());
        }
      }
    );
  }
  public void onDocumentUnloaded(
  IARControlEventsOnDocumentUnloadedEvent evt) throws IOException{
    /*set the statusbar text to empty string*/
    java.awt.EventQueue.invokeLater(
      new Runnable() {
           public void run() {
             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 ArcGIS API for Java. Adding listeners from the *java.awt.event* package, such as *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*.

In addition, these custom events are not fired from within Java's event dispatch thread. Whenever you wish to change the state of a pure Java GUI component from within one of these events, be sure to do so via java.awt.EventQueue's *invokeLater* method. On the other hand, the ArcGIS components do not run in Java's event dispatch thread. Because of this, you can change their state directly from within any of the custom ArcGIS events. For example, you could call *Map.refresh* from within an *ITransformEventsListener* without using *invokeLater*.

C++ API versus Visual C++ API: Do not confuse the C++ API with the Visual C++ COM API. If developing only on Windows, the Visual C++ COM API has advantages over the pure C++ API. The C++ API is primarily aimed at UNIX and cross-platform development.

C++ is an object-oriented programming language that evolved in the mid-1980s from its predecessor, C. C++ is endowed with many features that 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, C#, and VB.NET code.
- Cross-platform compatibility—Visual Basic, Visual C++, VB.NET, and C# are currently used primarily on the Windows platform. C++ and Java are inherently more cross-platform.
- Prior familiarity—If you 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

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 are 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*—This class encapsulates the *BSTR* data type.
- *CComVariant*—This 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 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;
}
```

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 identification is performed by comparing the pointer values of IUnknown. *The 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 its 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 C++ (and 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);
```

However, if you do have a value that you want 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 caller's 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 shown 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;
}
```

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 identity is performed by comparing the pointer values of IUnknown.*The smart pointers will perform necessary QI and comparison when using the "==" operator.*

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 the interface pointer is nonzero before calling AddRef.
  if (*pVal)
    *pVal->AddRef();

  return S_OK;
}
```

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.

When using a smart pointer to receive an object 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);

  // Initialize ipBar Smart pointer from Foo1.
  IBarPtr ipBar;
  ipFoo1->get_Bar(&ipBar);

  // The "&" de-reference will call Release on ipBar.
  // ipBar is then repopulated with a new instance of IBar.
  ipFoo2->get_Bar(&ipBar);
}
// ipBar goes out of scope, and the smart pointer destructor calls Release.
```

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—that is, types that are not smart types with overloaded cast operators.

```
// Perform QI of IUnknown.
IUnknownPtr ipUnk = ipSmartPointer;
// Ensure we use IUnknown* constructor of CComVariant.
CComVariant myVar2(ipUnk.GetInterfacePtr());
```

Here are some suggestions for a naming convention.These help identify the variable's usage and type and thus 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 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

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);
```

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 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. Click the C/C++ tab and click General from the Category dialog box. In the Debug info dialog box, click 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 (click the Eyeglasses button or press 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. 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 once some condition is reached (for example, 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 click the Condition button to display a dialog box in which you specify the break point condition.

Preloading DLLs

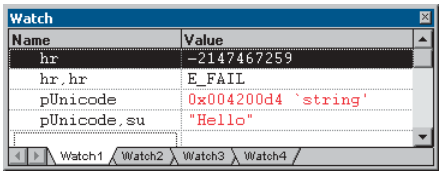
You can preload DLLs that you want to debug before executing the program. This allows you to set break points up front rather than wait until the DLL has been loaded during program execution. (Click Project, click Settings, click Debug, click Category, then click Additional DLLs.) Then, click in the list area below to add any DLLs you want 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.



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. Useful 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 to use the C++ API, a basic understanding of how COM objects work is necessary to use ArcObjects. If you are unfamiliar with the ArcObjects framework, the first two sections of this chapter, 'The Microsoft Component Object Model' and 'Developing with ArcObjects', are recommended reading.

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 ArcGISDevHelpVC.chm.

Getting help

For help on the objects used in this API (interfaces, classes, and so on), refer to the ArcGIS Developer Help for C++ that is installed with ArcGIS Engine. If you accepted the default installation options, the help system can be accessed as follows:

Windows

Go to Start menu > Programs > ArcGIS > Developer Help > C++ Help. Or you can open the C++ version of ArcGIS Developer Help from your ArcGIS Engine installation directory by navigating to \DeveloperKit\Help\COM and double-clicking ArcGISDevHelpVC.chm.

Solaris and Linux

Run <install location>/developerkit/viewArcGISDevHelpC++.

GETTING STARTED WITH C++ AND ARCOBJECTS

The C++ API can be a powerful tool for ArcObjects programming; however, there are some steps to take to get started with it. All of the documents mentioned in this section are in the help system as individual pages and can be found through the index. The goal of this document is to provide a single work flow that takes you through the steps of choosing a development environment, leaving you with a basic understanding of how to use the C++ API. Before you begin, you will want to make sure that your desired application is one that can be completed with the C++ API. Make sure your plans fall within the limitations of the C++ API, which can be found in the 'Limitations' section in this chapter.

For general information on consuming and extending with the APIs, see the discussion in Chapter 2 ,'ArcGIS software architecture', and in particular the section 'ArcGIS application programming interfaces'.

Pre-ArcObjects C++ configuration steps

Make sure that you are using a supported product and platform for the C++ API by checking the supported products and platforms. You should also verify that your C++ compiler is among those supported.

This section is intended to give you a starting point for learning the API. However, its purpose is not to teach you how to set up a Windows, Solaris, or Linux machine for C++ development, and it assumes that you can already compile a simple C++ program on your computer. The following steps provide a quick check to determine if your computer is at this point. For each of them, you will use this code snippet:

```
#include <iostream>
int main(int argc, char** argv)
{
 std::cerr << "Hello world" << std::endl;
}
```

- **Windows**:

1. Open your development environment of choice (either Visual Studio 6 or Visual Studio .NET 2003 [7.1]).
2. The way you create your new project will depend on the IDE you are using.

   In VS6:

   a. Start a new Win32 Console Application called configtest. Create it as an empty project.

   b. Add to the project a new C++ source file named configtest, and paste in the code from above.

In VS .NET 2003:

a. You want a simple C++ program, so use the wizard that you will use for your C++ API development: 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. This is available from the downloads page at *http://msdn.microsoft.com/academic/default.aspx*. Once you have installed the wizard, open a new Visual C++ project of the type C/C++ Console Application and name it configtest.

b. Replace the contents of the existing configtest.cpp file with the code above.

3. Press Ctrl+F5, and click Yes to build the .exe file. A window should open that says *Hello world.*

4. If you had trouble with any of the steps above, your system is not yet configured for C++ programming, and you should refer to your system documentation or system support personnel to get it set up.

- **Solaris and Linux**:

1. Open your text editor and paste the code snippet from above. Save it as configtest.cpp.

2. At the command prompt, type one of the following:

   - *Solaris*: "CC -o configtest configtest.cpp" (without the quotes)
   - *Linux*: "g++ -o configtest configtest.cpp" (without the quotes)

3. A new file, configtest, will be created. Run it by typing "./configtest" (again without the quotes).

4. *Hello world* will be displayed.

5. If you had trouble with any of the steps above, your system is not yet configured for C++ programming, and you should refer to your system documentation or system support personnel to get it set up.

Platform configuration

1. First you will need to pick your development platform: Windows, Solaris, or Linux. This will be a decision based on your ArcObjects programming intentions, as well as your experience.

- **Windows**: If you are planning to write command-line ArcObjects applications and most of your programming experience is on Windows, you should use Windows as your platform. Your code will still be cross-platform; you will just need to recompile it on Linux and Solaris.

  However, if you plan to write an application with the Motif or GTK widget ArcGIS controls, you will not be able to develop on Windows as those controls are only available on UNIX. To develop with the ArcGIS controls on Windows, you will need to use another API and its controls, such as the Visual C++ COM API and the COM controls.

You have a choice of development and compiler options for Windows, and your next step is going to be reading the documentation on the supported development environment you will be using. Your options are:

- Visual Studio 6 and the Visual Studio 6.0 C++ compiler
- Visual Studio .NET 2003 and the Visual Studio .NET 2003 C++ compiler (7.1)
- Using Visual Studio command-line tools (nmake) with either the Visual Studio 6.0 or .NET 2003 (7.1) compiler

Once you have read that documentation, continue with Step 2.

- **Solaris**: If you are planning to write ArcGIS control applications, or if you plan to write command-line ArcObjects applications, and you are most familiar with Solaris, you should use Solaris as your platform. Your command-line code will still be cross-platform: you will just need to recompile it on Linux and Windows.

  On Solaris you will be developing with the make utility and the Sun WorkShop (Forte) 6 update 2 C++ compiler. However, your development steps will depend on whether you are writing a command-line application or a control application. If you would like to write a Solaris control application, you will decide if you are going to write a Motif ArcGIS control application or a GTK ArcGIS control application. Both have benefits and disadvantages, as documented in 'Choosing between Motif and GTK'.

  Your next step is to read the documentation pertaining to the development environment you will be using:

  - Command-line applications with the make utility
  - Motif ArcGIS control applications with the make utility
  - GTK ArcGIS control applications with the make utility

  Once you have read that documentation, continue with Step 2.

- **Linux**: If you are planning to write ArcGIS control applications, or if you plan to write command-line ArcObjects applications, and you are most familiar with Linux, you should use Linux as your platform. Your command-line code will still be cross-platform: you will just need to recompile it on Solaris and Windows.

  On Linux you will be developing with the make utility and the GCC version 3.2 C++ compiler. However, your development steps will depend on whether you are writing a command-line application or a control application. If you would like to write a Linux control application, you will decide if you are going to write a Motif ArcGIS control application or a GTK ArcGIS control application. Both have benefits and disadvantages, as documented in 'Choosing between Motif and GTK'.

  Your next step is to read the documentation pertaining to the development environment you will be using:

  - Command-line applications with the make utility
  - Motif ArcGIS control applications with the make utility

- GTK ArcGIS control applications with the make utility

Once you have read that documentation, continue with Step 2.

2. Now that you know what platform you will be developing on, install the ArcGIS Engine Developer Kit on that platform, if you have not done so already. Make sure to register your ArcGIS Engine!
3. Before beginning with a new API, it can be helpful to have some idea why things might *not* be working. Some of the common configuration mistakes are covered in the 'Troubleshooting' section in this chapter.
4. You should now have a computer configured for C++ development and know how to set up, compile, and run ArcObjects C++ code in your development environment of choice. The next step is to become familiar with ArcObjects and the C++ API.

Getting to work

1. If you are programming on either Solaris or Linux, you now need to initialize your ArcGIS Engine.

   You should also be aware that if a C++ API application crashes on Solaris or Linux, you need to run mwcleanup, as discussed in 'Solaris and Linux post-crash cleanup' in this chapter.
2. ArcObjects is based on the Microsoft Component Object Model, or COM, and successful ArcObjects programming requires a basic understanding of COM. Read 'Building an understanding of COM and ArcObjects' earlier in this chapter for information on developing COM knowledge and the ArcGIS system. The resources in that document are only a brief introduction. A more complete understanding of the ArcObjects system can be developed through the overviews in the Library Reference for each of the ArcGIS libraries and by taking ESRI courses targeted at ArcObjects programmers.
3. Now you will begin to get into the details of the ArcObjects C++ API. To get started, step through one of these walkthroughs. The walkthrough you complete should reflect your ArcObjects C++ goals:
   - For command-line application programmers (Windows, Solaris, or Linux): See the section 'Building a command-line C++ application' in Chapter 6.
   - For ArcGIS control application programmers (Solaris or Linux): See the section 'Building applications with C++ and Motif widgets' in Chapter 6.
4. You should now be able to write your own applications using the ArcObjects C++ API. If you are programming on either Solaris or Linux, or if you are programming on Windows using the Visual Studio command-line tools, you will need to write a makefile to go with your application. Template makefiles have been provided for your use. Select the makefile appropriate for your developer environment and modify it as discussed in the platform configuration document you were directed to above.
   - Windows command-line programming with nmake
   - Solaris command-line programming with make

- Linux command-line programming with make
- Solaris Motif ArcGIS control programming with make
- Linux Motif ArcGIS control programming with make
- Solaris GTK ArcGIS control programming with make
- Linux GTK ArcGIS control programming with make

Similarly, if you wrote code on one platform and now want to transfer it to another, simply copy your code to the supported platform you want to transfer it to, select and modify the appropriate makefile from above, and compile and run the application on the new platform.

5. This introduction has not covered every piece of the C++ API. Another tool provided for you are the samples. To further your C++ API knowledge, as well as your understanding of the ArcObjects framework, read through and try some of the samples. For information on how to use the C++ samples, see the online Help topic 'How to Use the C++ Samples', and to find samples for C++ you can use the samples query page in the online Help.

   Although not all of the samples are written in C++, they are still useful for learning the ArcObjects framework and the recommended method calls to use in development. You may find code written in Visual Basic that does a task you would like to do in C++. See the section 'Converting from Visual Basic to C++' in this chapter for some tips on converting VB code to C++.

6. There are important C++ API topics that have not been covered in this introduction. As you need additional information, refer to the following documents:

   - Some of the functions specific to the C++ API have been introduced to you through the walkthroughs and samples, but the complete documentation on them is in the 'Recommended function usage' section in this chapter.
   - Recommended C++ programming techniques:
     - Smart types
     - Naming conventions
     - Debugging tips in Developer Studio
   - Recommended ArcObjects programming practices:
     - Cocreating an object with a smart pointer reference after the smart pointer is declared
     - ESRI System interfaces
     - Raw pointers in function signatures
     - Return ArcObjects from functions
     - Inline query interface
     - Replicating the functionality of instanceof (Java) or TypeOf (Visual Basic)
     - Getting a feature's field values from type VARIANT

- Programming with Motif:
  - Getting started with Motif programming
  - Motif ArcGIS control programming
- Programming with GTK:
  - Getting started with GTK programming
  - GTK ArcGIS control programming
- Event handling on ArcGIS controls
- Walkthroughs discussing additional areas of the C++ API can be found in the developer help system under Development Environments > C++ > Walkthroughs. They include how to write custom commands, tools, and events.
- Error handling

INITIALIZING THE SOLARIS AND LINUX ARCGIS ENGINE

With your machine ready for C++ development, there is only a single step to prepare for ArcGIS Engine development, and that is to source the <Your Installation directory>/arcgis/init_engine.sh (or .csh, depending on your shell of choice). If you prefer, that can be done in your shell's RC file (.cshrc or .bashrc, for example). Otherwise, you must source that file once per shell.

If you have the engine installed and registered, but you are getting the error message "Error: Could not open include file <ArcSDK.h>.", try sourcing the init_engine script.

Supported compilers

If your GCC compiler on Linux has not been installed in a standard location, some of your compiled applications may not be able to find libstdc++.so at runtime. In this case, you will need to add this library's directory (usually <gcc-install-dir>lib) to your LD_LIBRARY_PATH environment variable.

On Windows, the Visual Studio 6.0 and Visual Studio .NET 2003 compilers are supported by the ArcGIS C++ API.

On Solaris, the Sun Workshop (Forte) 6 update 2 compiler is supported.

On Linux, the supported compiler is GCC version 3.2.

Development environments

You are also able to mix and match by coding in one development environment and compiling in another. For example, you can write code in Visual Studio but compile and build it via a script utilizing the command-line tools. Keep in mind, however, that Motif and GTK code will not compile or run on Windows.

As an ArcGIS C++ API developer, you can choose any development environment as long as you use a supported compiler. However, if you want to use an integrated development environment (IDE) on Windows, either Visual Studio 6.0 or Visual Studio .NET 2003 (7.1) is recommended. If you don't want to use an IDE, you can write your code in any text editor and compile it from the command line. If you choose this option, it is recommended that you use the Windows nmake utility, the Solaris make utility, or the Linux make utility, accessed from the corresponding command prompt. Your choice of development environment depends entirely on your personal preference and the tools available. This section focuses on each of these options, as well as the option of using Motif or GTK in your Solaris and Linux applications.

ARCGIS DEVELOPMENT IN THE VISUAL STUDIO 6.0 IDE

Setting 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. Click File > New > Win32 Console Application, type in the name of the project, and choose its location.

The ArcSDK.h and ArcGIS Engine OLB files are installed in the <ArcGIS install directory>\include\CPPAPI and <ArcGIS install directory>\com folders, respectively.

Next, set the required project options. On the C/C++ tab of the Project Menu > Settings dialog box, click Preprocessor from the Category combo box. 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 semicolon.

Go to the Preprocessor Definitions text box and type "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 to add some files to the project; these will eventually contain your code. Select the type of file you want 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!

Compiling your application

To compile an ArcGIS Engine application in Visual Studio 6.0, press F7 or click Build > Build YourApplicationName.exe.

Running 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 clicking Build > Execute YourApplicationName.exe or by pressing Ctrl+F5. To run the application in debug mode, click Build > Start Debug > Go, or press F5.

ARCGIS DEVELOPMENT IN THE VISUAL STUDIO .NET 2003 IDE

Setting 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. This is available from the downloads page at *http://msdn.microsoft.com/academic/*

default.aspx. Once you have installed the wizard, click New > Project > Visual C++ Projects > C/C++ Console Application.

Now you are ready to proceed with your project options. First, add some additional include directories. Do this by clicking Project Menu > Properties > C/C++ folder > 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 semicolon! You can also click the ellipses to add new directories.

The ArcSDK.h and ArcGIS Engine .olb files are installed in the <ArcGIS install directory>\include\CPPAPI and <ArcGIS install directory>\com folders, respectively.

Next, click Preprocessor and in the Preprocessor Definitions text box type "ESRI_WINDOWS" to define the ESRI_WINDOWS symbol. You can also click the ellipses to define symbols.

Now you are ready to write your code. Don't forget to start by including ArcSDK.h!

Compiling your application

To compile an ArcGIS Engine application in Visual Studio .NET 2003, click Build > Build Solution.

Running 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 combo box.

Finally, run the application by clicking Debug > Start Without Debugging or by pressing Ctrl+F5. If you want to run the application in debug mode, click Debug > Start, or press F5.

ARCGIS DEVELOPMENT WITH NMAKE AND THE WINDOWS COMMAND PROMPT

Setting 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 choose 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.

The vcvars32.bat file's default location is Program Files\Microsoft Visual Studio\VC98\Bin.

Accessing 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, 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.

1. Open a command prompt and use the "cd" command to change to the directory containing vcvars32.bat.
2. Type "vcvars32.bat" to run the batch file.
3. For development, use the "cd" command to change 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 file with any necessary parameters.

- Create a batch file to run vcvars32.bat for you.
  1. Navigate to the directory in which you want to store the batch file.
  2. Right-click in the directory and click 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 click Edit.
  5. Find vcvars32.bat, right-click the file, and click 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 want to develop from 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, use the "cd" command to change 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 file with any necessary parameters.

Accessing 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.

Setting 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 template Makefile.Windows provided with ArcGIS Engine in ArcGIS Developer Help. Refer to the next section for details on this sample file.

These steps assume that you have installed to the default location. 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 template makefile

The template makefile, Makefile.Windows, can be found in ArcGIS Developer Help under Development Environments > C++ > Makefiles.

As a convenience, a template makefile—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 single code and header files, my_application.cpp and my_application.h, and produces an executable that takes in a single file at runtime.

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.

1. Throughout the makefile, update the program name, currently 'basic_sample', to reflect your application name. In this example, my_application is the program 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
```

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.

2. The compiler options outlined in Steps 2 through 4 above have been set for you; however, you need to complete Step 1 yourself to prepare the template for use in your applications.

```
…
# 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
```

This line was also shown in Step 1 to illustrate the update of the program name.

3. Provide dependencies lists for your application.

```
…
# 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!

Compiling your application

Once Makefile.Windows is ready to compile your application, you can compile from the command line by typing "nmake /f Makefile.Windows".

Running your application

You can either invoke your application directly or through the makefile. If you choose to invoke it directly, you will need to provide command-line parameters from the command line. To use the makefile to run an ArcGIS Engine command-line application, you must set up the command-line parameters in the makefile. 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 it from the command line by typing "nmake /f Makefile.Windows run".

If your GCC compiler on Linux has not been installed in a standard location, some of your compiled applications may not be able to find libstdc++.so at runtime. In this case, you will need to add this library's directory (usually <gcc-install-dir>/lib) to your LD_LIBRARY_PATH environment variable.

ArcGIS DEVELOPMENT WITH MAKE AND THE SOLARIS/LINUX COMMAND PROMPT

Setting up a compiler for use from the command prompt

Sun WorkShop (Forte) 6 update 2 (CC) for Solaris; GCC version 3.2 (g++) for Linux

Initializing ArcGIS Engine

With your machine ready for C++ development, there is only a single step to prepare for ArcGIS Engine development, and that is to source the arcgis/init_engine.sh (or .csh, depending on your shell of choice). If you prefer, that can be done in your shell's RC file (.cshrc or .bashrc, for example). Otherwise, you must source that file once per shell.

Setting up your application

If desired, you can utilize the template Makefile.Solaris or Makefile.Linux provided with ArcGIS Engine in ArcGIS Developer Help. Refer to the next section for details on these sample files.

Open your favorite text editor and begin writing your code. Use a makefile to set the following include directories, library options, and compiler options.

If desired, you can utilize the template makefiles provided with ArcGIS Engine. Makefile.Solaris provides a starting point for Solaris command-line applications, and Makefile.Linux will get you started with Linux command-line applications. Refer to the next section for details on these templates.

Below, *$(ARCENGINEHOME)* is used to refer to the root directory of your ArcGIS Engine install and should be defined once you have sourced arcgis/init_engine.sh (or .csh). For examples of each of these steps, see the template makefiles: Makefile.Solaris and Makefile.Linux.

1. Use the -I compiler option to add some additional include directories:
   - $(ARCENGINEHOME)/include
   - (Linux) /usr/X11R6/include (or the correct version for your installation)
2. Use the -L linker option to specify some additional library directories:
   - $(ARCENGINEHOME)/bin
   - (Linux) /usr/X11R6/lib (or the correct version for your installation)
3. Use the -l linker option link against the arcsdk library.
4. Use the -D compiler option to define the ESRI_UNIX symbol to direct the compiler to read the UNIX support headers from within ArcSDK.h.

Customizing the templates Makefile.Solaris and Makefile.Linux

The template makefiles, Makefile.Solaris and Makefile.Linux, can be found in ArcGIS Developer Help under Development Environments > C++ > Makefiles.

As a convenience, template makefiles, named Makefile.Solaris for Solaris command-line applications and Makefile.Linux for Linux command-line applications, are included with ArcGIS Engine for your use. The following steps highlight the specific areas of those files that must be customized for you to use them in your development process. The modifications shown are based on an application that is written in a single code and a single header file, my_application.cpp and my_application.h, and produces an executable that takes in a single file at runtime.

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.

1. Throughout the makefile, update the program name, currently 'basic_sample', to reflect your application name. In this example, the program name is my_application.

```
# Set up the program name
PROGRAM = my_application
 ...
# Program name updates - source list
CXXSOURCES = my_application.cpp
 ...
# Program name updates - objects, dependencies, and compilation commands
my_application.o: my_application.cpp my_application.h
$(CXX) $(CXXFLAGS) -c -o my_application.o my_application.cpp
```

2. Update the dependencies list for your application. This line was also shown above to illustrate the update of the program name. However, it may also involve adding additional parameters and lists if the application you are writing is broken up into more files.

```
 ...
# Program name updates - objects, dependencies, and compilation commands
my_application.o: my_application.cpp my_application.h
$(CXX) $(CXXFLAGS) -c -o my_application.o my_application.cpp
```

With your makefile prepared, you are ready to write your code. Don't forget to start by including ArcSDK.h!

Compiling your application

Once Makefile.Solaris or Makefile.Linux is ready to compile your application, you can compile from the command line by typing "make -f Makefile.Solaris" or "make -f Makefile.Linux", as appropriate.

Running your application

You can either invoke your application directly or through the makefile. If you choose to invoke it directly, you will need to provide command-line parameters from the command line. To use the makefile to run an ArcGIS Engine command-line application, you must set up the command-line parameters in the makefile. 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 = /mycomputer/data/inputfile
...
# Setting up a run target
run:
   $(PROGRAM) $(INPUT)
```

Once Makefile.Solaris or Makefile.Linux is ready for use with your application, you will be able to run from the command line by typing "make -f Makefile.Solaris run" or "make -f Makefile.Linux run", as appropriate.

ARCGIS MOTIF DEVELOPMENT WITH MAKE AND THE SOLARIS/LINUX COMMAND PROMPT

If your GCC compiler on Linux has not been installed in a standard location, some of your compiled applications may not be able to find libstdc++.so at runtime. In this case, you will need to add this library's directory (usually <gcc-install-dir>/lib) to your LD_LIBRARY_PATH environment variable.

Setting up a compiler for use from the command prompt

Sun WorkShop (Forte) 6 update 2 (CC) for Solaris; GCC version 3.2 (g++) for Linux

Initializing ArcGIS Engine

With your machine ready for C++ development, there is only a single step to prepare for ArcGIS Engine development, and that is to source the arcgis/init_engine.sh (or .csh, depending on your shell of choice). If you prefer, that can be done in your shell's RC file (.cshrc or .bashrc, for example). Otherwise, you must source that file once per shell.

If desired, you can utilize the template Makefile.SolarisMotif or Makefile.LinuxMotif provided with ArcGIS Engine in ArcGIS Developer Help. Refer to the next section for details on these sample files.

Setting up your application

Setting up an ArcGIS Engine C++ Motif application is exactly like setting up an ArcGIS Engine C++ application—there are only a few additional libraries to add to the makefile. For those of you familiar with ArcGIS Engine C++ programming, you will only need to add motifctl, aoctl, Xm, Xt, and X11 as libraries to link against (that is, add them with the -l flag, as shown in Step 3 below). If you are not familiar with ArcGIS Engine C++ programming, all the steps you need to take in preparing your application are discussed below.

Open your favorite text editor and begin writing your code. Use a makefile to set the following include directories, library options, and compiler options.

If desired, you can utilize the template makefiles provided with ArcGIS Engine. Makefile.SolarisMotif provides a starting point for Solaris Motif applications, and Makefile.LinuxMotif will get you started with Linux Motif applications. Refer to the next section for details on these templates.

Below, *$(ARCENGINEHOME)* is used to refer to the root directory of your ArcGIS Engine install and should be defined once you have sourced arcgis/init_engine.sh (or .csh). For examples of each of these steps, see the template makefiles: Makefile.SolarisMotif and Makefile.LinuxMotif.

1. Use the -I compiler option to add some additional include directories:
   - $(ARCENGINEHOME)/include
   - (Linux) /usr/X11R6/include (or the correct version for your installation)
2. Use the -L linker option to specify some additional library directories:
   - $(ARCENGINEHOME)/bin
   - (Linux) /usr/X11R6/lib (or the correct version for your installation)
3. Use the -l linker option link against some libraries:
   - (Linux) pthread
   - Xm
   - Xt
   - X11
   - arcsdk
   - motifctl
   - aoctl
4. Use the -D compiler option to define the ESRI_UNIX symbol to direct the compiler to read the UNIX support headers from within ArcSDK.h.

Customizing the templates Makefile.SolarisMotif and Makefile.LinuxMotif

The template makefiles, Makefile.SolarisMotif and Makefile.LinuxMotif, can be found in ArcGIS Developer Help under Development Environments > C++ > Makefiles.

As a convenience, template makefiles, named Makefile.SolarisMotif for Solaris Motif applications and Makefile.LinuxMotif for Linux Motif applications, are included with ArcGIS Engine for your use. The following steps highlight the specific areas of those files that must be customized for you to use them in your development process. The modifications shown are based on an application that is written in a single code and a single header file, my_application.cpp and my_application.h, and produces an executable that takes in a single file at runtime.

1. Throughout the makefile, update the program name, currently 'motif_sample', to reflect your application name. In this example, the program name is my_application.

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.

```
# Set up the program name
PROGRAM = my_application.exe
 ...
```

```
# Program name updates - source list
CXXSOURCES = my_application.cpp
 ...
# Program name updates - objects, dependencies, and compilation commands
my_application.o: my_application.cpp my_application.h
$(CXX) $(CXXFLAGS) -c -o my_application.o my_application.cpp
```

2. Update the dependencies list for your application. This line was also shown above to illustrate the update of the program name. However, it may also involve adding additional parameters and lists if the application you are writing is broken up into more files.

```
 ...
# Program name updates - objects, dependencies, and compilation commands
my_application.o: my_application.cpp my_application.h
$(CXX) $(CXXFLAGS) -c -o my_application.o my_application.cpp
```

With your makefile prepared, you are ready to write your code. Don't forget to start by including ArcSDK.h!

Compiling your application

Once Makefile.SolarisMotif or Makefile.LinuxMotif is ready to compile your application, you can compile from the command line by typing "make -f Makefile.SolarisMotif" or "make -f Makefile.LinuxMotif", as appropriate.

Running your application

You can either invoke your application directly or through the makefile. If you choose to invoke it directly, you will need to provide command-line parameters from the command line. To use the makefile to run your ArcGIS Engine application, type "make -f Makefile.SolarisMotif run" or "make -f Makefile.LinuxMotif run", as appropriate.

ArcGIS GTK DEVELOPMENT WITH MAKE AND THE SOLARIS/LINUX COMMAND PROMPT

Setting up a compiler for use from the command prompt

If your GCC compiler on Linux has not been installed in a standard location, some of your compiled applications may not be able to find libstdc++.so at runtime. In this case, you will need to add this library's directory (usually <gcc-install-dir>/lib) to your LD_LIBRARY_PATH environment variable.

Sun WorkShop (Forte) 6 update 2 (CC) for Solaris; GCC version 3.2 (g++) for Linux

For Solaris GTK users, we recommend that you download the GNOME desktop from http://wwws.sun.com/software/star/gnome/index.html.

Initializing ArcGIS Engine

With your machine ready for C++ development, there is only a single step to prepare for ArcGIS Engine development, and that is to source the arcgis/init_engine.sh (or .csh, depending on your shell of choice). If you prefer, that can be done in your shell's RC file (.cshrc or .bashrc, for example). Otherwise, you must source that file once per shell.

If desired, you can utilize template makefiles provided with the ArcGIS Engine. Makefile.SolarisGTK provides a starting point for Solaris GTK applications, and Makefile.LinuxGTK will get you started with Linux GTK applications. Refer to the next section for details on these templates.

Setting up your application

Setting up an ArcGIS Engine C++ GTK application is like setting up an ArcGIS Engine C++ application. There are only a few additional libraries, includes, and flags to add to the makefile. For those of you familiar with ArcGIS Engine C++ programming, you will need to add gtkctl and aoctl as libraries to link against (that is, add them with the -l flag, as shown in Step 3 below). In addition, you will need to add the GTK CFLAGS and LDFLAGS, which will be generated by a package management script called pkg-config. If you are not familiar with ArcGIS Engine C++ programming, all the steps you need to take in preparing your application are discussed below.

Open your favorite text editor and begin writing your code. Use a makefile to set the following include directories, library options, and compiler options.

Below, *$(ARCENGINEHOME)* is used to refer to the root directory of your ArcGIS Engine install and should be defined once you have sourced arcgis/init_engine.sh (or .csh). For examples of each of these steps, see the template makefiles: Makefile.SolarisGTK and Makefile.LinuxGTK.

1. Use the -I compiler option to add some additional include directories:

- $(ARCENGINEHOME)/include
- (Linux) /usr/X11R6/include (or the correct version for your installation)

2. Use the -L linker option to specify some additional library directories:

- $(ARCENGINEHOME)/bin
- (Linux) /usr/X11R6/lib (or the correct version for your installation)

3. Use the -l linker option link against some libraries:

- arcsdk
- gtkctl
- aoctl

4. Use the -D compiler option to define the ESRI_UNIX symbol to direct the compiler to read the UNIX support headers from within ArcSDK.h.

5. Generate GTK compiler and linker arguments with pkg-config:

- $(shell pkg-config gtk+-2.0 —cflags)
- $(shell pkg-config gtk+-2.0 —libs)

Customizing the templates Makefile.SolarisGTK and Makefile.LinuxGTK

The template makefiles, Makefile.SolarisGTK and Makefile.LinuxGTK, can be found in ArcGIS Developer Help under Development Environments > C++ > Makefiles.

As a convenience, template makefiles, named Makefile.SolarisGTK for Solaris GTK applications and Makefile.LinuxGTK for Linux GTK applications, are included with ArcGIS Engine for your use. The following steps highlight the specific areas of those files that must be customized for you to use them in your development process. The modifications shown are based on an application that is written in a single code and a single header file, my_application.cpp and my_application.h, and produces an executable that takes in a single file at runtime.

1. Throughout the makefile, update the program name, currently 'gtk_sample', to reflect your application name. In this example, the program name is my_application.

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.

```
# Set up the program name
PROGRAM = my_application
 ...
# Program name updates - source list
CXXSOURCES = my_application.cpp
 ...
# Program name updates - objects, dependencies, and compilation
commands
my_application.o: my_application.cpp my_application.h
$(CXX) $(CXXFLAGS) -c -o my_application.o my_application.cpp
```

2. Update the dependencies list for your application. This line was also shown above to illustrate the update of the program name. However, it may also involve adding additional parameters and lists if the application you are writing is broken up into more files.

```
 ...
# Program name updates - objects, dependencies, and compilation
commands
my_application.o: my_application.cpp my_application.h
$(CXX) $(CXXFLAGS) -c -o my_application.o my_application.cpp
```

With your makefile prepared, you are ready to write your code. Don't forget to start by including ArcSDK.h!

Compiling your application

Once Makefile.SolarisGTK or Makefile.LinuxGTK is ready to compile your application, you can compile from the command line by typing "make -f Makefile.SolarisGTK" or "make -f Makefile.LinuxGTK", as appropriate.

Running your application

You can either invoke your application directly or through the makefile. If you choose to invoke it directly, you will need to provide command-line parameters from the command line. To use the makefile to run your ArcGIS Engine application, type "make -f Makefile.SolarisGTK run" or "make -f Makefile.LinuxGTK run", as appropriate.

For the sake of simplicity the code snippets given don't always check HRESULTs, although as a developer you should always do so.

RECOMMENDED FUNCTION USAGE

The C++ API provides its own implementation of some functions. Below are the names and descriptions of the general API functions and examples of their use. For additional information on the Motif- and GTK-specific functions, see the sections 'Motif ArcGIS control programming' and 'GTK ArcGIS control programming', respectively.

General API functions (described below)

- AoInitialize
- AoUninitialize
- AoExit
- AoCreateObject
- AoAllocBSTR
- AoFreeBSTR
- AoToolbarAddCommand
- AoToolbarAddTool

Motif-specific API functions

- MwCtlAppMainLoop
- MwCtlGetInterface

GTK-specific API functions

- gtk_axctl_new
- gtk_axctl_get_interface
- gtk_axctl_initialize_message_queue

AoInitialize, AoUninitialize, and AoExit

- *AoInitialize*—used where CoInitialize would be used in COM programming.

  ```
  extern "C" HRESULT AoInitialize(LPVOID pvReserved);
  ```

  This function initializes ArcGIS Engine. The initialization must be done prior to ArcObjects being used and in addition to the use of the *IAoInitialize* interface, which handles licensing for the application.

- *AoUninitialize*—used where CoUninitialize would be used in COM programming.

  ```
  extern "C" void AoUninitialize(void);
  ```

  This function uninitializes ArcGIS Engine.

- *AoExit*—used where exit would be used in non-ArcObjects code, as well as where 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 elements.

The following example illustrates how the three functions discussed above should be used within an application.

In this code, both AoInitialize *and* IAoInitialize *are used. These are not the same thing:* AoInitialize *is the API call discussed above, while* IAoInitialize *is an ArcObjects interface used in licensing.*

Note that IAoInitialize ***must*** *be 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

- *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 and AoFreeBSTR

- *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);
```

AoToolbarAddCommand and AoToolbarAddTool

- AoToolbarAddCommand replaces IToolbar->AddItem when adding a custom command. The custom command class must inherit from CAoCommandBase.

```
extern "C" HRESULT AoToolbarAddCommand(IToolbarControl* pToolbarControl,
                                       CAoCommandBase* commandBase,
                                       enum esriCommandStyles style);
```

- *AoToolbarAddTool* replaces IToolbar->AddItem when adding a custom tool. The custom tool class must inherit from *CAoToolBase*.

```
extern "C" RESULT AoToolbarAddTool(IToolbarControl* pToolbarControl,
                                   CAoToolBase* commandBase,
                                   enum esriCommandStyles style);
```

For examples on the use of *AoToolbarAddCommand* and *AoToolbarAddTool*, see 'Creating custom commands and tools'.

ArcObjects C++ practices

For the sake of simplicity, the code snippets given don't always check HRESULTs, although as a developer you should always do so.

The following are some recommendations for programming with the ArcGIS C++ API.

Getting 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 it is assumed 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;
  }
}
```

Cocreating 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

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(…);
```

Replicating 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(pDataset);
   if (ipFeatureDataset != 0)
   {
      // Use IFeatureDataset methods.
   }
   else
   {
      IRasterDataset2Ptr ipRasterDataset(pDataset);
      if (ipRasterDataset!= 0)
       {
         // Use IRasterDataset2 methods.
       }
   }
}
```

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 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 want to return and assign it to the pointer you were passed.

```
HRESULT GetTinWorkspace(char* path, ITinWorkspace** ppTinWorkspace)
{
  if (!ppTinWorkspace)
    return E_POINTER;
  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;
}
```

Notice the call to AddRef. *This is required to ensure that resources are managed properly.*

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 types similar to some of these exist; 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—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—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 is 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);
```

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);
```

ERROR HANDLING

You might notice that the samples and scenarios do not follow the Good Error Handling practices outlined here. This is done simply to increase code readability since error checking is not the focus of those bits of code.

COM methods return an HRESULT to signify the success or failure of a call, as discussed in the 'Developing with ArcObjects' section earlier 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 HRESULTs 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.

```
// 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;
}
```

- 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, you 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;
    }
```

CONVERTING FROM VISUAL BASIC TO C++

While many samples are provided in multiple languages, the bit of code you would like to use might only be available in Visual Basic. If you encounter this, here are some tips to convert that code into C++. However, these are not complete bits of code, and variables are not always defined. They are just examples of how to translate a pattern in VB into C++ syntax.

A few notes on variable names:

VB	C++	Notes

Dim pWsFact As _
IWorkspaceFactory
pWsFact = New _
RasterWorkspaceFactor
	IWorkspaceFactoryPtr ipWsFact (CLSID_Raster WorkspaceFactory); // if creating at declaration
	OR
	IWorkspaceFactoryPtr ipWsFact;// if creating later than defined
	ipWsFact.CreateInstance (CLSID_Raster WorkspaceFactory);
		Coding practice depends on when the instance is created.

Set pWs = pWsFact._ OpenFromFile(sDir, 0)	ipWsFact->OpenFromFill(CComBSTR(sDir), 0, &ipWs);	C++ requires an explicit cast to CComBSTR, and ipWs must be created previously and passed in (even if still NULL).
Set pRasterDataset = _ pWs.OpenRaster_ Dataset(sFile)	((IRasterWorkspacePtr) ipWksp)->OpenRaster Dataset(CComBSTR(sFile), ppRasterDataset);	Inline query interfacem to IRasterWorkspace from IWorkspace.

Set pFieldEdit = pField	IFieldEditPtr ipFieldEdit(ipField); // ipFieldEdit not yet declared
	OR
	ipFieldEdit = ipField; // ipFieldEdit previously declared
		Coding practice depends on where the instance is given its value.

pFieldEdit.Name = "Shape"	ipFieldEdit->put_Name (CComBSTR(shape FieldName));	Use "put_" for assignments.
Set pFieldEdit.Geometry_ Def = pGeomDef	ipFieldEdit->putref_GeometryDef(ipGeomDef);	If VB uses "Set" in the assignment, you will use "putref_" in C++.
fcount = pfields.FieldCount	ipFields->get_FieldCount(&lFieldCount);	Use "get_" to retrieve information. Notice you will need to pass in a reference.
if (TypeOf m_ipDisplay_ Feedback Is INewMulti_ PointFeedback) Then	INewMultiPointFeedbackPtr ipNewMultiPoint Feedback (m_ipDisplayFeedback); if (ipNewMultiPointFeedback != 0)	

- "p" prefix—variable is a pointer.
- "pp" prefix—variable is a pointer to a pointer.
- "ip" prefix—variable is a smart pointer.

TROUBLESHOOTING

Problems finding or opening the 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. If you installed to the default directory, it should be either *\Program Files\ArcGIS\include\CPPAPI* (for Windows) or */arcgis/include* (for Linux and Solaris). If it is correct and it is still not working, make sure the file is in that directory.

If it is in the correct location and you are using either Solaris or Linux, the next thing to try is running arcgis/init_engine.sh (or .csh). The message may mean that $ARCENGINEHOME/include isn't in the compiler's include path. In the samples, our makefiles typically take care of adding this for you. However, if $ARCENGINEHOME isn't set, it will fail, and it is set in the arcgis/init_engine script.

Error: Please define either ESRI_WINDOWS or ESRI_UNIX

You need to inform the compiler of which set of header files to use. On Windows, you need to define the ESRI_WINDOWS symbol. On Solaris and Linux, you need to define the ESRI_UNIX symbol. If you are using a makefile (either with nmake on Windows or make on Solaris or Linux), you will set this with the "D" compiler flag. In Visual Studio 6.0, you set it on the C/C++ tab of the Project Menu > Settings dialog box by selecting "Preprocessor" from the Category combo box and adding "ESRI_WINDOWS" to the "Preprocessor Definitions" textbox. In Visual Studio .NET 2003, the symbol is defined in the Project Menu > Properties dialog box in the C/C++ folder by clicking Preprocessor and adding "ESRI_WINDOWS" to the Preprocessor Definitions textbox.

Windows-specific errors

- **Cannot open type library file esriSystem.olb**

  This Windows error means that your sample is not compiling because it cannot open an ESRI OLB file. Make sure that you have the correct argument for the COM include directory, which is ArcGIS\Com in a default installation. 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.**

  On Windows, if samples or your own code compiles but fails to run, make sure you registered your ArcGIS Engine. Run the SoftwareAuthorization tool found in Start > All Programs > ArcGIS and follow its prompts.

Solaris- and Linux-specific errors

- Error: No valid runtime license was found.

  This means that you have not registered your ArcGIS Engine. Run the SoftwareAuthorization tool found in arcgis/authorizeSoftware and follow its prompts.

- fatal: libarcsdk.so: open failed: No such file or directory (Solaris)

  OR

  /usr/bin/ld: cannot find -larcsdk (Linux)

  This tells you that $ARCENGINEHOME/bin isn't in the LD_LIBRARY_PATH, which is set up when you run the arcgis/init_engine.sh (or .csh) script. Run the init_engine script.

LIMITATIONS

When using the C++ API, only command-line applications are cross-platform.

On Solaris and Linux, controls in the form of Motif widgets have been provided and GUI applications can be built. The Motif applications will not run on Windows. However, GUI applications can be built on Windows 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 of the help system.

SOLARIS AND LINUX POST-CRASH CLEANUP

If an application does not exit cleanly, it is possible that processes will remain. To tidy up these processes, use the mwcleanup utility. Typing "mwcleanup" at the command line will kill all running ArcGIS applications and clean up all X properties.

CHOOSING BETWEEN MOTIF AND GTK

If you would like to write a control application on Solaris or Linux, you have two widget sets to choose from: Motif and GTK. One major factor in whether you should use Motif or GTK will be your programming experience: if you are already familiar with one, that will make your ArcObjects experience easier. In addition, each has both benefits and disadvantages.

Pros	
Motif	**GTK**
multiplatform availability	multiplatform availability
traditional UNIX look and feel	widely used for newer projects
well supported and documented	fairly easy to program
preinstalled on Solaris	standard part of most Linux distributions
long-established industry standard	active development

Cons	
Motif	**GTK**
difficult to program	noncentralized support
little active development	documentation is lacking
support for Linux is lacking	support for Solaris is quite new
dated design	

MOTIF PROGRAMMING

Getting started with Motif programming

Motif widget ArcGIS controls have been provided for C++ developers. To use them, you must understand some basics of Motif programming. This is not by any means a complete resource; the variety of Motif widgets and their resources, which you have available to you as a programmer, are not discussed here. However, this should give you a place to start figuring out the Motif-specific bits of the C++ API and samples.

Seven steps of Motif programming

When writing a Motif program, there are seven steps that need to be done.

1. Initialize the Motif Toolkit.
2. Create the widgets.

3. Manage the widgets.
4. Implement event listening and callback functions for widgets.
5. Display the widgets.
6. Begin the event handling loop.
7. Shut down the application.

To illustrate each of these steps, you will create a simple Motif application.

Motif Program: Simple PushButton

This application will be a single button that displays the number of times the button was clicked repeatedly to cerr (for example, if double-clicked it will display "2 clicks"). Start a new file that will be your program. Here that file will be called pbExample.cpp.

To use the Motif Toolkit you will need to include the Motif header file. The Motif PushButton widget you will be using requires Xm/PushB.h (each widget's header file includes Xm/Xm.h, but it is good practice to include it anyway). For the display of "pushed", you will need to include *iostream*. Xm/Protocols.h will be used for the callbacks. You will also need a *main* function. Set these up in your new file, so that it looks like this:

```
#include <iostream>
#include <Xm/Xm.h>
#include <Xm/PushB.h>
#include <Xm/Protocols.h>

int main(int argc, char* argv[])
{
  return 0;
}
```

at the top of your new file.

Step 1: Initialize the Motif Toolkit

Your first initialization step is to set the language procedure for Xt. Do so by calling XtSetLanguageProc. Then you will initialize the Motif Toolkit with a call to XtVaAppInitialize. This call does a few things: connects the application to the display; gets any standard command-line arguments; sets up resources; and creates and returns the top-level window widget, which will be the parent of all other widgets in this applicaiton.

A deeper understanding of these calls is not necessary to using Motif as it is used in samples. However, if you would like further information on the parameters passed into either function, see the *Motif Programming Manual* resource listed at the end of this topic.

*In ArcGIS Engine applications, you **must** use* AoInitialize *as well, placing the call before any ArcObjects usage.*

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.

```
int main(int argc, char* argv[])
{
  XtSetLanguageProc(NULL, NULL, NULL);
  XtAppContext app_context;
  Widget topLevel = XtVaAppInitialize(&app_context, "XApplication",
                                      NULL, 0, &argc, argv, NULL, NULL);
```

```
  return 0;
}
```

Step 2: Create the widgets

With the toolkit initialized, you can create the single widget you are using in this application. There are two ways to create widgets:

- Using a function specific to the particular widget:

  `XmCreatePushButton()`
- Using a function for generic widget creation (and sometimes managing it at the same time):

  `XtVaCreateWidget()`
  `XtVaCreateManagedWidget()`

Although you will see both in the C++ samples for Motif, here you will use the second method, but will not manage the widget at its creation (simply to separate that step for the purpose of this introduction).

```
int main(int argc, char* argv[])
{
  XtSetLanguageProc(NULL, NULL, NULL);
  XtAppContext app_context;
  Widget topLevel = XtVaAppInitialize(&app_context, "XApplication",
                                      NULL, 0, &argc, argv, NULL, NULL);

  XmString label = XmStringCreateLocalized("Push the button.");
  Widget button = XtVaCreateWidget("button",
                                   xmPushButtonWidgetClass,   topLevel,
                                    XmNlabelString,           label,
                                    NULL);
  XmStringFree(label);

  return 0;
}
```

The parameters have the following roles:

- *"button"*—the name of the widget in the resource database, which can be used for specifications in a resource file. If a label is not provided, it will act as the widget's label.
- *xmPushButtonWidgetClass*—the class of the widget to be created. For example, to create an ArcGIS control widget, you would give *mwCtlWidgetClass*.
- *topLevel*—the parent of the widget, which must be a manager widget that was already created.
- *XmNlabelString, label, NULL*—resource settings. For more information on the resources available on different widgets and how to use them, see the Motif reference at the end of this topic. Some common resources used in the C++ samples for Motif are those to set the size and placement of the widget. In addition, there is a custom resource that goes with the ArcGIS controls: *MwNprogID*.

Step 3: Manage the widgets

For the child to appear in the application, it must be managed. A single call, *XtManageChild*, will do this for you, placing control of the widget in its parent's hands. This must be done for each widget. Place this line of code after the call to *XmStringFree*, as shown.

```
XmStringFree(label);

XtManageChild(button);
```

Note: Even if a child is managed, it will not appear if its parent is not managed.

Step 4: Implement event listening and callback functions for widgets

You now have a button, but for that to be useful you must hook it to some functionality. Widgets are attached to behavior at certain events through special callback functions. You can add callbacks to a widget after it is created and either before or after it is managed. As done in most of the C++ samples for Motif, here you add the callback before it is managed.

```
XmStringFree(label);
XtAddCallback(button, XmNactivateCallback, ClickCallback, NULL);

XtManageChild(button);
```

These parameters have the following roles:

- *button*—the widget to add the callback to.
- *XmNactivateCallback*—the callback resource, defined by Motif to correspond to certain events. Here you will pick activation of the button. For other options, see the Motif reference at the end of this topic.
- *ClickCallback*—pointer to the function to call on the event.
- *NULL*—Client data to pass into the callback function. Here there is no data the function will need, so NULL is passed.

Callback functions

For the callback to work, you must implement the function that is being called on the event. Callbacks have a specific function signature, as follows:

```
void myCallbackName(Widget w, XtPointer client_data, XtPointer call_data)
```

The parameters are:

- *w*—the widget that was activated for this callback to be called.
- *client_data*—any data passed into the function, as indicated in the last parameter of *XtAddCallback*.
- *call_data*—a structure containing data specific to the type of widget with which the callback is associated.

Make sure that the data passed to client_data will be in scope later when the callback routine is executed.

For this example, place this function after *main* in pbExample.cpp, and have it print to cerr the number of repeated clicks on this button (for example, "2 clicks" if double-clicked). Remember to also place a forward declaration of it before *main*.

To get the number of clicks, you must access the call_data, which has the type *XmPushButtonCallbackStruct*. To do so, you will need to cast the *XtPointer* to this callback struct type, then you can get the integer member click_count.

```
void ClickCallback(Widget w, XtPointer client_data, XtPointer call_data)
{
  XmPushButtonCallbackStruct* data = (XmPushButtonCallbackStruct*) call_data;
  std::cerr << data->click_count << std::endl;
}
```

Step 5: Display the widgets

Your application's look and functionality are now written, but to create the actual window for your widget you need to call *XtRealizeWidget* right before you start the event loop, as you will do in the next step. This call is only needed on the top-level widget, which will then recursively realize the rest of the widgets.

```
XtManageChild(button);

XtRealizeWidget(topLevel);
```

Step 6: Begin the event handling loop

The next step for this application is to turn control of the application over to Xt, which will manage the events. This code will idle until a user generates an event.

```
XtRealizeWidget(topLevel);
XtAppMainLoop(app_context);
```

For ArcGIS control programming, MwCtlAppMainLoop ***must*** *be used in the place of* XtAppMainLoop*, as you will see in ArcGIS C++ code.*

For ArcGIS control programming, MwCtlAppMainLoop *must* be used in the place of XtAppMainLoop, as you will see in ArcGIS C++ code.

Step 7: Shut down the application

As mentioned above, this application will run indefinitely unless it receives an event that tells it to do otherwise. To allow proper shutdown of the application, you will handle the window manager message that the window is going to be closed.

After the button is managed and before the event loop is started in *main*, you will listen for that window manager message:

```
XtManageChild(button);

Atom wm_delete_window = XmInternAtom(XtDisplay(g_topLevel),
                                     "WM_DELETE_WINDOW", FALSE);
XmAddWMProtocolCallback(g_topLevel, wm_delete_window, CloseAppCallback,
                        NULL);
```

You will also provide the callback that has the application close when that event is heard. As for the other callback, make sure you place a forward declaration before *main*.

For ArcGIS Engine C++ programming, you ***must*** *use* AoExit *in place of* exit.*You* ***must*** *call* AoUninitialize *before shutting down the application with* AoExit.

```
void CloseAppCallback(Widget w, XtPointer client_data, XtPointer call_data)
{
  exit(0);
}
```

Trying it out

To compile your Motif program, you will need to link against libraries for Motif, Xt, and X11, in that order. If you are programming on Solaris, you will compile with the Sun Workshop (Forte), and if you are programming on Linux you will use GCC.

```
CC pbExample.cpp -o pbExample -lXm -lXt -lX11
```

Run the program:

```
./pbExample
```

As you click the button, you will see counts appear in your terminal window.

Now that you have a feeling for Motif programming, look at the C++ samples for Motif, and see these steps applied there.

Additional resources

- Heller, Dan, Paula M. Ferguson, and David Brennan. *Motif Programming Manual* (The Definitive Guides to the X Window System, Volume 6A) 2nd Edition. O'Reilly & Associates. 1994.

Motif ArcGIS control programming

The Solaris and Linux SDK also provides a set of Motif widgets that may be used to embed the ArcGIS controls in a Motif application. Due to the limitations of Motif, this code will only run on Solaris and Linux systems. If visual components are needed on Windows, you will need to use Visual C++, the COM API, and ActiveX controls.

Header files

To use the Motif ArcGIS controls, you will need to include both the ArcGIS Engine header file, ArcSDK.h, and the ArcGIS Engine Motif controls header file, Ao/AoMotifControls.h.

Control types and their details

Control	MwNprogID	Interface Type
Globe Control	AoPROGID_GlobeControl	IGlobeControl
Map Control	AoPROGID_MapControl	IMapControl3
PageLayout Control	AoPROGID_PageLayoutControl	IPageLayoutControl
Reader Control	AoPROGID_ReaderControl	IARControl
Scene Control	AoPROGID_SceneControl	ISceneControl
TOC Control	AoPROGID_TOCControl	ITOCControl
Toolbar Control	AoPROGID_ToolbarControl	IToolbarControl

API functions and arguments

- For each ArcGIS Engine Control interface, there is a smart pointer defined for you. Instead of *IMapControl3**, you can use *IMapControl3Ptr*.
- *MwCtlAppMainLoop* replaces *XtAppMainLoop*.

```
extern "C" void MwCtlAppMainLoop(XtAppContext app);
```

- *MwCtlGetInterface* is used to get the control's interface pointer.

```
HRESULT MwCtlGetInterface(Widget w, IUnknown** ppUnk);
```

- *mwCtlWidgetClass*—argument to give the widget class for all ActiveX control widgets
- *MwNprogID*—argument to pair with *AoPROGID_<ControlName>Control*

Control widget creation example

The following example demonstrates creating a map control that fills the entire Motif mainForm (defined outside this code snippet) and retrieves a smart pointer for the control. The bold sections highlight functions and resources discussed above. The *#define* statements are necessary to prevent type name conflicts between X and ArcGIS Engine.

```
// Motif Headers
#define String       esriXString
#define Cursor       esriXCursor
#define Object       esriXObject
#define ObjectClass   esriXObjectClass
#include <Xm/Xm.h>
#include <Xm/MainW.h>
#include <Xm/Form.h>
#include <Xm/Protocols.h>
#undef String
#undef Cursor
#undef Object
#undef ObjectClass

// ArcObjects Headers
// Engine
#include <ArcSDK.h>
// Controls
#include <Ao/AoMotifControls.h>

int main(int argc, char* argv[])
{
  ...

 Widget mapWidget = XtVaCreateWidget("mapWidget",
                                     mwCtlWidgetClass,    mainForm,
                                     XmNtopAttachment,      XmATTACH_FORM,
                                     XmNbottomAttachment,   XmATTACH_FORM,
                                     XmNleftAttachment,     XmATTACH_FORM,
                                     XmNrightAttachment,    XmATTACH_FORM,
                                     MwNprogID,         AoPROGID_MapControl,
                                     NULL);
 IMapControl3Ptr ipMapControl;
 HRESULT hr = MwCtlGetInterface(mapWidget, (IUnknown**)&ipMapControl);

  ...
}
```

Setting the size of a control widget

If you would like to set your control widget to have a specific size, you will need to do this after the widget itself is created. Do *not* use XmNheight and XmNwidth in the XtVaCreateWidget section. Due to internal limitations, that will cause undetermined behavior in the controls. Instead, set the size of your widget after it is created with a call to XtVaSetValues. For example, to set the above mapControl to be 200x200, add the line:

```
XtVaSetValues(mapWidget, XmNheight, 200, XmNwidth, 200, NULL);
```

after the call to XtVaCreateWidget.

GTK PROGRAMMING

Getting started with GTK programming

GIMP Toolkit (GTK) widget ArcGIS controls have been provided for C++ developers. To use them, you must understand some basics of GTK programming. This is not by any means a complete resource; the variety of GTK widgets and their resources, which you have available to you as a programmer, are not discussed here. However, this should give you a place to start figuring out the GTK-specific bits of the C++ API and samples.

Seven steps of GTK programming

When writing a GTK program, there are seven steps that need to be done.

1. Initialize GTK.
2. Create the widgets.
3. Place the widgets.
4. Implement event listening and callback functions for widgets.
5. Show the widgets.
6. Begin the event handling loop.
7. Shut down the application.

To illustrate each of these steps, you will create a simple GTK application.

GTK Program: Simple PushButton

This application will be a single button that displays which button on the mouse was used to click on the button widget. Start a new file that will be your program. Here that file will be called pbExample.cpp.

To use GTK you will need to include the GTK header file: gtk/gtk.h. For the display of the number of the button clicked, you will need to include iostream. You will also need a main function. Set these up in your new file, so that it looks like this:

```
#include <iostream>
#include <gtk/gtk.h>

int main(int argc, char* argv[])
{
  return 0;
}
```

In ArcGIS Engine applications, you must use AoInitialize as well, placing the call before any ArcObjects usage.

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.

Step 1: Initialize GTK

Initialize GTK with a call to gtk_init(&argc, &argv);. This call does a few things: sets up resources, initializes everything needed to work with GTK, and parses some command-line options.

```
int main(int argc, char* argv[])
{
 gtk_init(&argc, &argv);

 return 0;
}
```

Step 2: Create the widgets

With the toolkit initialized, you can create the single widget you are using in this application. First you will need to create the window in which you will place the button. Then you will create the button itself with gtk_button_new_with_label, since you want a labeled button in this example.

```
int main(int argc, char* argv[])
{
 gtk_init(&argc, &argv);

 GtkWidget *window, *button;

 window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
 button = gtk_button_new_with_label("Push the button.");

 return 0;
}
```

Step 3: Place the widgets

For the widgets to show up, they must be added to the main window as follows:

```
button = gtk_button_new_with_label("Push the button.");
gtk_container_add(GTK_CONTAINER(window), button);
```

To use multiple widgets in an application, they must be packed. This is done with horizonal or vertical boxes or panes. Some of the calls related to packing include gtk_box_pack_start(), gtk_hbox_new(), gtk_vpaned_new(), and gtk_paned_add1(). Since this example only uses a single widget, this step is not needed. For additional information on it, see either the reference at the end or the samples.

Step 4: Implement event listening and callback functions for widgets

You now have a button, but for that to be useful you must hook it to some functionality. Widgets are attached to behavior at certain events through special callback functions. You can add callbacks to a widget after it is created.

```
button = gtk_button_new_with_label("Push the button.");
g_signal_connect(G_OBJECT(button), "button_press_event",
    G_CALLBACK(ClickCallback), NULL);
```

These parameters have the following roles:

- button—the widget that will give off the signal you are watching for.

- "button_press_event"—the signal to respond to. Here you will listen for the button being pressed. For other options, see the GTK reference at the end of this topic.
- ClickCallback—the function to call when the signal is received.
- NULL—data to pass into the callback function. Here there is no data the function will need, so NULL is passed.

Callback functions

For the callback to work, you must implement the function that is being called on the event. Callbacks will generally follow the following function signature:

```
void callback_func(GtkWidget *widget, gpointer callback_data);
```

However, as in this case, you will sometimes want additional information about the event. To get that information, the following function signature can be used:

```
void ClickCallback(GtkWidget *widget, GdkEventButton *event, gpointer
  callback_data);
```

The parameters are:

- widget—the widget that gave the signal
- event—tells what button press or release event triggered the function
- callback_data—any data passed into the function, as indicated in the last parameter of g_signal_connect

Make sure that the data passed to callback_data will be in scope later when the callback routine is executed.

For this example, place this function after main in pbExample.cpp, and have it print to cerr which mouse button was used to click (for example, "button: 1"). Remember to also place a forward declaration of it before main.

To get the button that was clicked, you must use the information in the GdkEventButton struct.

```
void ClickCallback(GtkWidget *widget, GdkEventButton *event,
   gpointer callback_data)
{
 // show which button was clicked
 std::cerr << "button pressed: " << event->button << std::endl;
}
```

Step 5: Show the widgets

Your application's look and functionality are now written, but to create the actual window for your widget you need to call show_all_ right before you start the event loop, as you will do in the next step. You can either call gtk_widget_show on each widget, or you can call gtk_widget_show_all on the top-level widget, which will then recursively realize the rest of the widgets.

```
g_signal_connect(G_OBJECT(button), "button_press_event",
   G_CALLBACK(ClickCallback), NULL);

gtk_widget_show_all(window);
```

Step 6: Begin the event handling loop

The next step for this application is to idle until a user generates an event.

```
gtk_widget_show_all(window);

gtk_main();
```

Step 7: Shut down the application

As mentioned above, this application will run indefinitely unless it receives an event that tells it to do otherwise. To allow proper shutdown of the application, you will handle the signals that tell you the window is going to be closed.

After the widgets are shown and before the event loop is started by gtk_main, you will listen for those signals:

```
gtk_widget_show_all(window);

g_signal_connect(G_OBJECT(window), "delete_event", G_CALLBACK(delete_event),
    NULL);
g_signal_connect(G_OBJECT(window), "destroy", G_CALLBACK(destroy_event), NULL);
```

You will also provide the callbacks DeleteEvent and DestroyEvent that have the application close when the signals are recieved. As for the other callback, make sure you place a forward declaration before main.

```
static void destroy_event(GtkWidget *widget, gpointer data)
{
 gtk_main_quit();
}

static gboolean delete_event(GtkWidget *widget, GdkEvent *event,
    gpointer data)
{
 return FALSE; // must return false to trigger destroy event for window
}
```

*For ArcGIS Engine C++ programming, you **must** call* AoExit *before returning. You **must** call* AoUninitialize *before shutting down the application with* AoExit.

Trying it out

To compile your GTK program, you will need to link against libraries for GTK, Xt, and X11, in that order. If you are programming on Solaris, you will compile with the Sun Workshop (Forte), and if you are programming on Linux you will use GCC.

Solaris:

CC pbExample.cpp -o pbExample ‘pkg-config gtk+-2.0 —cflags —libs‘

Linux:

g++ pbExample.cpp -o pbExample ‘pkg-config gtk+-2.0 —cflags —libs‘

Run the program:

```
./pbExample
```

As you click the button, you will see the number of the button you pressed appear in your terminal window.

Now that you have a feeling for GTK programming, look at the C++ samples for GTK, and see these steps applied there.

Additional resources

- www.gtk.org

GTK ArcGIS control programming

The Solaris and Linux SDK also provides a set of GTK widgets that may be used to embed the ArcGIS controls in a GTK application. Due to the limitations of GTK, this code will only run on Solaris and Linux systems. If visual components are needed on Windows, you will need to use Visual C++, the COM API, and ActiveX controls.

Header files

For an ArcGIS GTK control application, you will need to include Ao/GtkControls.h in addition to your ArcGIS Engine and GTK includes (ArcSDK.h and gtk/gtk.h, respectively).

Control types and their details

Control	ProgID	Interface Type
Globe Control	AoPROGID_GlobeControl	IGlobeControl
Map Control	AoPROGID_MapControl	IMapControl3
PageLayout Control	AoPROGID_PageLayoutControl	IPageLayoutControl
Reader Control	AoPROGID_ReaderControl	IARControl
Scene Control	AoPROGID_SceneControl	ISceneControl
TOC Control	AoPROGID_TOCControl	ITOCControl
Toolbar Control	AoPROGID_ToolbarControl	IToolbarControl

API functions and arguments

- For each ArcGIS Engine Control interface, there is a smart pointer defined for you. Instead of IMapControl3*, you can use IMapControl3Ptr.
- gtk_axctl_new is used to create the GTK ArcGIS Control Widget.

```
GtkWidget *gtk_axctl_new(ControlDataPtr cd);
```

- gtk_axctl_get_interface is used to get the control's interface pointer.

```
HRESULT gtk_axctl_get_interface(GtkWidget *axctl, IUnknown **ppUnk);
```

- gtk_axctl_initialize_message_queue is used to enable MainWin message delivery before starting the main GTK loop with gtk_main.

```
void gtk_axctl_initialize_message_queue();
```

Control widget creation example

The following example demonstrates creating and placing a map control that fills the entire GTK form window (defined outside this code snippet) and retrieves a smart pointer for the control. The bold sections highlight functions and resources discussed above.

```
// ArcObjects Headers
// Engine
#include <ArcSDK.h>
// Controls
#include <Ao/AoGtkControls.h>
```

```
int main(int argc, char* argv[])
{
  ...

  GtkWidget* mapWidget;
  IMapControl3Ptr ipMapControl;

  mapWidget = gtk_axctl_new(AoPROGID_MapControl);
  gtk_axctl_get_interface(mapWidget, (IUnknown **)&ipMapControl);
  gtk_widget_set_size_request(mapWidget, -1, -1);
  gtk_container_add(GTK_CONTAINER(window), mapWidget);

  ...
}
```

EVENT HANDLING ON ARCGIS CONTROLS

Events occur any time there is interaction with an application during its runtime. When a new map is loaded, when the mouse moves inside the control window, or when a keyboard button is pressed are examples of when events occur, and at such times the information regarding the event is reported. If the application is listening for events, it will execute code associated with the event that has occurred.

To make an application that responds to events on the controls without having a tool or command selected, you will not only need to tell the application what to do when such events are received, but will also have to tell the computer to be listening for and reacting to such events. This document details how to listen for the events, but does not go into detail on how to write your event class. For help on writing an event class, see 'Writing a MapControl Event class' and 'Writing a Transform Event class' in the developer help system under Development Environments > C++ > Walkthroughs.

Although there are also Motif and GTK events, interacting with ArcObjects (such as popup menus) through those events is not recommended. Instead, you should interact through the ArcObjects event interfaces.

If, however, you want to use a tool or a command, you will not need to listen for events. Custom tools and commands interact with the controls through their own functions, which are called upon similar events. See 'Creating a custom command using AoBaseCommand' or 'Creating a custom tool using AoBaseTool' in the developer help system under Development Environments > C++ > Walkthroughs.

Events on the controls

To handle control events you will need to write a class that inherits from the events with which you want to interact. For the main events on a control, which implement *IDispatch*, your C++ API event classes will inherit from a helper interface. Listening for these events is then simplified. The events this applies to are listed in the table below. For all other event types, you will implement a class that inherits from the ArcObjects interface with the events you want to listen for. Then you will listen for the events as outlined under 'Other ArcObjects events—Implemented with custom interfaces' below.

Once you have implemented your class, you will need to have your application listen for the events. To do so, you will use *IEventHelper*, which passes information between your application and the events, handling reference counting for you.

Initiating the listening process

Once you have written your event class, you will need to set up your application to listen for the events. This is a two-step process:

1. *IEventListenerHelper->Startup*: sets up a link between the sink (your object, such as a *MapControl*) and the listener helper object.

   This step will need to be done before realizing the top-level widget and entering the main application loop.

2. *IEventListenerHelper->AdviseEvents*: sets up a link between the listener helper object and the source of the events—for example, *IActiveViewEvents*. Since the main events on the controls are implemented using *IDispatch*, you will pass in NULL for the interface to listen to, as shown in the example below.

Where this step is taken will depend on the type of object on which you want to listen. If you are listening on an object that does not change—for example, a *MapControl*—you can do this right after you call Startup on the events. If, however, you want to listen to an object that will change during a single run of the application, such as the ActiveView, you will want to make this call each time that object changes. For example, to listen to events on a *MapControl*'s ActiveView, you would want to make this call in the *MapControl*'s *OnMapReplaced* event. This does mean that you would also need to implement the *MapControl*'s main events.

Ending the listening process

For your application to exit cleanly, you will need to stop your application from listening for events before you shut it down. This is again a two-step process:

1. *IEventListenerHelper->Shutdown*: removes the link between the sink (your object, such as a *MapControl*) and the listener helper object.
2. *IEventListenerHelper->UnadviseEvents*: removes the link between the listener helper object and the source of the events—for example, *IActiveViewEvents*.

Since these steps will only need to be taken when the application is closed, both calls will be made from the callback that listens for the application to be closed, before *IAoInitialize*'s shutdown is called and before the application is uninitialized.

Main control events—Implemented with IDispatch

To handle the main events of the controls, write a class that inherits from a helper interface that implements the events you are going to listen for. For example, to listen for *IPageLayoutControlEvents*, write a class that inherits from *IPageLayoutControlEventsHelper*. For further implementation details, see the walkthrough 'Writing a MapControl Event class' in the developer help system under Development Environments > C++ > Walkthroughs. You will then listen for the events by using the helper interface *IEventListenerHelper*.

Include files

To use these events, you will need to include Ao/AoControls.h in your event class's header file.

Implementation details

To initiate listening for control-specific events, create an instance of your event class. Then create a new instance of *IEventListenerHelper* with a listener class ID that matches your event type. Call *IEventListenerHelper->Startup*, then *IEventListenerHelper->AdviseEvents*. Notice that NULL is passed into the AdviseEvents call because a dispatch interface is being used. Your application is now set up to listen for the events you have implemented in your event class.

Before shutting down the application, you will need to call *IEventsListenerHelper-> UnadviseEvents*, then *IEventsListenerHelper->Shutdown*. Remember to delete any associated global variables, such as the instance of your class, and to set any global interface pointers to 0.

Control	Inherit from	Event Listener Class ID
Globe control	IGlobeControlEventHelper	CLSID_GlobeControlEventsListener
Map control	IMapControlEvents2Helper	CLSID_MapControlEvents2Listener
PageLayout control	IPageLayoutControlEventsHelper	CLSID_PageLayoutControlEventsListener
Reader control	IARControlEventsHelper	CLSID_ARControlEventsListener
Scene control	ISceneControlEventsHelper	CLSID_SceneControlEventsListener
Table of Contents control	ITOCControlEventsHelper	CLSID_TOCControlEventsListener
Toolbar control	IToolbarControlEventsHelper	CLSID_ToolbarControlEventsListener

Example

```
// Set up event listening
//   PageLayoutControlEvents is a custom class that inherits from
//     IPageLayoutControlEventsHelper
//   g_pageLayoutEvents is a global PageLayoutControlEvents*
//   g_ipPageLayoutControlEventHelper is a global IEventListenerHelperPtr
//   g_ipPageLayoutControl is a global IPageLayoutControlPtr
g_pageLayoutEvents = new PageLayoutControlEvents();
g_ipPageLayoutControlEventHelper.CreateInstance(CLSID_PageLayoutControlEventsListener);
g_ipPageLayoutControlEventHelper->Startup(
  static_cast<IPageLayoutControlEventsHelper*> (g_pageLayoutEvents));
g_ipPageLayoutControlEventHelper->AdviseEvents(g_ipPageLayoutControl, NULL);

...

// You will need to clean up the events. This is done when the application
//    is given the signal to close.
g_ipPageLayoutControlEventHelper->UnadviseEvents();
g_ipPageLayoutControlEventHelper->Shutdown();
g_ipPageLayoutControlEventHelper = 0;
delete g_pageLayoutEvents;
// Shutdown application with IAoInitialize's Shutdown
```

Other ArcObjects events—Implemented with custom interfaces

To listen for the other events that the controls send, write a class that inherits from the ArcObjects interface that implements the events with which you want to interact. For example, to listen for *ICustomizeDialogEvents*, write a class that inherits from that interface. For further implementation details, see the walkthrough 'Writing a Transform Event class' in the developer help system under Development Environments > C++ > Walkthroughs.

Include files

To use these events, you will need to include Ao/AoControls.h in your event class's header file.

Implementation details

To initiate listening for these events, you first need to get the interface on which to Advise (in the example below, *IActiveView*). Create an instance of your event class. Call both *IEventListenerHelper->Startup* and *IEventListenerHelper->AdviseEvents*. For these events, you will need to inform AdviseEvents of the UID of the type of events to listen for, as these events use custom interfaces and not a dispatch implementation.

Before shutting down the application, you will need to call *IEventsListenerHelper->UnadviseEvents*, then *IEventsListenerHelper->Shutdown*. Remember to delete any associated global variables, such as the instance of your class, and to set any global interface pointers to 0.

- The listener class IDs of the other events will be in the form:

  `CLSID_<Coclass>Listener`

  For example, to listen for ActiveViewEvents, use *CLSID_ActiveViewEventsListener* as the class ID.

- Inherit from the ArcObjects interface that implements the events—*IActiveViewEvents*, for example.

Example

```
// Set up event listening
// ActiveViewEvents is a custom class that inherits from IActiveViewEvents
// g_activeviewEvents is a global ActiveViewEvents*
// g_ipActViewEventHelper is a global IEventListenerHelperPtr
// g_ipMapControl is a global IMapControl2Ptr
g_activeviewEvents = new ActiveViewEvents();
g_ipActViewEventHelper.CreateInstance(CLSID_ActiveViewEventsListener);
g_ipActViewEventHelper->Startup(static_cast<ActiveViewEvents*>
  (g_activeviewEvents));
CComBSTR bsGUID;
::StringFromIID(IID_IActiveViewEvents, &bsGUID);
IUIDPtr ipUID(CLSID_UID);
ipUID->put_Value(CComVariant(bsGUID));
IActiveViewPtr ipActiveView;
g_ipMapControl->get_ActiveView(&ipActiveView);
g_ipActViewEventHelper->AdviseEvents(ipActiveView, ipUID);

...

// You will need to clean up the events. This is done when the application
   is given the signal to close.
g_ipActViewEventHelper->UnadviseEvents();
g_ipActViewEventHelper->Shutdown();
g_ipActViewEventHelper = 0;
delete activeviewEvents;
// Shut down application with IAoInitialize's Shutdown
```

CREATING CUSTOM COMMANDS AND TOOLS

With ArcGIS Engine, you can write custom commands and tools to add to your applications. They allow you to easily add custom functionality to your ArcGIS control applications without having to listen for all the events on the controls. To create a new command or tool, you will use the *esriSystemUI ICommand* interface. Through the *ICommand* interface, you will be able to set the properties and behavior for your command or tool. Some of the properties that you can set through the *ICommand* interface are the command's name, bitmap, caption, category, statusbar message, tooltip, enabled state, and checked state. It also defines the action taken when your command is clicked. For a custom tool, you will also use the *esriSystemUI ITool* interface. Through the *ITool* interface, you will be able to specify what cursor to use; what to do when the mouse button is pressed, released, or double-clicked; what to do when the mouse is moved; what to do when a key is pressed down or released; and what to do when a screen display in the application is refreshed.

Since commands are a GUI tool to interact with ArcGIS controls, C++ custom commands will only work on Solaris and Linux. If you want to write a command or tool to plug into ArcMap or ArcCatalog, you will need to use a different language such as Visual C++.

Your custom command will be a button or menu that performs a simple action when clicked or selected. If you only want to have an action performed when the custom toolbar button is clicked (as, for example, a zoom to full extent command), you only need to write a custom command. To create custom commands with the C++ API, there is a helper class for *ICommand* that your command class will inherit from: *CAoCommandBase*. This is defined in arcgis/include/Ao/AoCommandBase.h and includes *AoToolbarAddCommand*, which you will use to place your custom command on a ToolbarControl.

Your custom tool will be a button or menu that interacts with the controls when it is selected. If you are looking to interact with the ArcGIS controls' display—for example, as the zoom in tool does—you will need to write a custom tool. To create custom tools with the C++ API, there is a helper class for *ICommand* and *ITool* that your tool class will inherit from: *CAoToolBase*. This is defined in arcgis/include/Ao/AoToolBase.h and includes *AoToolbarAddTool*, which you will use to place your custom tool on a ToolbarControl.

When you write your custom command or tool class, you will first stub out and implement all members of *ICommand* (for commands and tools) and *ITool* (for tools). This document does not go into detail on how to write your command or tool class. For help on writing the class, see 'Creating a custom command using AoBaseCommand' or 'Creating a custom tool using AoBaseTool', which can be found in the developer help system under Development Environments > C++ > Walkthroughs.

Once your command or tool class has been implemented, you can add it into your application that has a toolbar. For the purposes of this document, ipToolbarControl will refer to your application's toolbar, which is already set up and placed earlier in the code. Your command class will be referred to as MyCommandClass, and your tool class will be referred to as MyToolClass.

1. Your first step will be to include the header file for your command or tool and update the makefile to reflect this addition.
2. Next create an instance of your command or tool.

```
// Create an instance of your command
MyCommandClass* myCommand = new MyCommandClass();

// Create an instance of your tool
MyToolClass* myTool = new MyToolClass();
```

3. Now place your command or tool onto your application's toolbar.

```
// Place the command onto the toolbar
AoToolbarAddCommand(ipToolbarControl, myCommand, esriCommandStyleIconOnly);

// Place the tool onto the toolbar
AoToolbarAddTool(ipToolbarControl, myTool, esriCommandStyleIconOnly);
```

   Your options for the third parameter to *AoToolbarAddCommand* and *AoToolbarAddTool* are the same as they are for adding built-in commands and tools to the toolbar: *esriCommandStyleIconOnly*, *esriCommandStyleTextOnly*, and *esriCommandStyleIconAndText*.

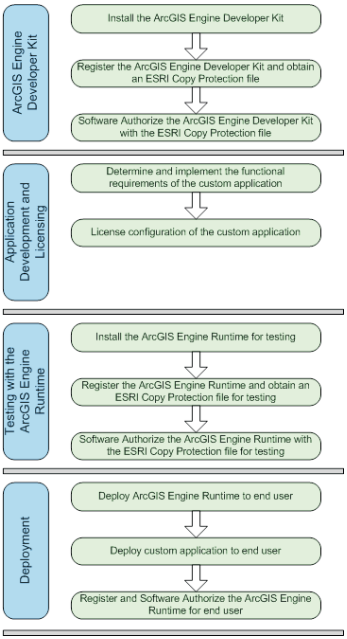
4. Compile your application. Your command or tool will be on the toolbar, ready for you to use it.

5 Licensing and deployment

Development of ArcGIS Engine applications cannot be undertaken in isolation from the testing and deployment of the final application. Prior to testing, ArcGIS Engine Runtime must be installed and the software authorized for use. Deployment of the application involves three separate processes: license initialization within the application, installation of the ArcGIS Engine Runtime software, and the authorization of its use.

This chapter details each of these processes and examines the licensing and deployment decisions that you, as the developer, must make prior to distributing your ArcGIS Engine applications.

This chapter examines the steps required to successfully install, authorize, build, and deploy an application with the ArcGIS Engine Developer Kit. The steps covered are installation, registration, and authorization of the ArcGIS Engine Developer Kit and ArcGIS Engine Runtime; license configuration of custom applications; testing custom applications with ArcGIS Engine Runtime; and deploying custom applications to a end users.



KEY DEFINITIONS

ArcGIS Engine Developer Kit: A component-based software developer product for building and deploying custom GIS and mapping applications.

ArcGIS Engine Runtime: An end user product that must be installed on any machine a custom application developed using the ArcGIS Engine Developer Kit is deployed to.

Licensing: The grant to a party of the right to use a software package or component.

Copy Protection: The mechanism used to license the ArcGIS Engine Developer Kit or ArcGIS Engine Runtime on a per-machine, single-use basis via an ESRI Copy Protection (*.ecp) file.

Software Authorization: Configuring the ArcGIS Engine Developer Kit or ArcGIS Engine Runtime with an ESRI Copy Protection (*.ecp) file to 'unlock' the underlying software components for use.

Deployment: The installation of a component or application to a target machine.

PRODUCT LICENSING

On installation of the ArcGIS Engine Developer Kit and ArcGIS Engine Runtime, you will need to read and agree to the ESRI Master License Agreement (MLA). The MLA defines the permitted use of the software. A copy of the MLA can be found in C:\Program Files\ArcGIS\License or at *http://www.esri.com/licenseagreement*.

The ArcGIS Engine Developer Kit is a single-use license product, meaning a license must be dedicated for each computer that has access to the software. With the ArcGIS Engine Developer Kit, you have the right to develop an unlimited number of applications on a single computer and deliver the applications to others. It is illegal to redistribute the ArcGIS Engine Developer Kit registration number or authorization file.

The ArcGIS Engine Runtime is also a single-use license, meaning it is intended for dedicated desktop computers. Each machine running a custom application developed with the ArcGIS Engine Developer Kit will require an ArcGIS Engine Runtime license or an ArcGIS Desktop license. The ArcGIS Engine Runtime license does not permit the ArcGIS Engine Runtime to be used for development purposes, nor can it be deployed on a server machine. The ArcGIS Engine Runtime setup can be redistributed, but the license and registration number cannot without authorization from ESRI. Any number of applications can use the same ArcGIS Engine, as long as they are all installed on the same machine.

RUNTIME LICENSING

There are two types of ArcGIS Engine Runtime licenses—product licenses and extension licenses. Product licenses include core ESRI software, such as ArcGIS Engine Runtime, ArcGIS Desktop licenses (ArcView, ArcEditor, ArcInfo), and ArcGIS Server. Extension licenses include additional ESRI products that build on those core licenses, such as the Spatial extension for ArcGIS Engine, the 3D Analyst extension, and the Network extension. This book deals solely with ArcGIS Engine-based development; therefore, ArcGIS Server product and extension licensing will be ignored. Since, as mentioned in Chapter 1, ArcGIS Engine Runtime is the platform on which ArcGIS Desktop is built, ArcGIS Desktop licensing may factor into your application development and is discussed here.

Licensing and deployment considerations for ArcGIS Server-based development are discussed in the ArcGIS Server Administrator and Developer Guide.

Custom applications that utilize core ArcGIS Engine features can be built to run against any or all of the following product runtime licenses:

- ArcGIS Engine Runtime
- ArcView
- ArcEditor
- ArcInfo

The ArcGIS controls, as part of ArcGIS Engine, also follow this runtime licensing model. MapControl, PageLayoutControl, ReaderControl, TOCControl, and ToolbarControl applications can utilize the ArcGIS Engine Runtime, ArcView, ArcEditor, or ArcInfo product licenses. Since GlobeControl and SceneControl extend core ArcGIS Engine functionality, they require a corresponding 3D extension license in addition to the core product license.

Applications that utilize extended ArcGIS Engine features require extension licenses that correspond to the ArcGIS product providing the core license. In other words, if your application initializes with an ArcGIS Engine Runtime license, any needed extension licenses must be ArcGIS Engine Runtime extensions, not ArcGIS Desktop extensions.

The extended ArcGIS Engine features and the extension runtime licenses that can be used to initialize them are:

- Multiuser enterprise geodatabase—ArcGIS Engine Runtime with Geodatabase Update extension or an ArcEditor or ArcInfo license.
- Visualization of data in 3D, GlobeControl, and SceneControl—ArcGIS Engine Runtime with 3D extension or an ArcGIS Desktop (ArcView, ArcEditor, or ArcInfo) license with the 3D Analyst extension.
- Creation and analysis of cell-based raster data—ArcGIS Engine Runtime with Spatial extension or an ArcGIS Desktop (ArcView, ArcEditor, or ArcInfo) license with the ArcGIS Spatial Analyst extension.
- Creation and management of network datasets and network-based spatial analysis—ArcGIS Engine Runtime with the Network extension or an ArcGIS Desktop (ArcView, ArcEditor, or ArcInfo) license with the ArcGIS Network Analyst extension.

For more detailed descriptions of the features provided by the ArcGIS Engine extensions, see Chapter 1, 'Introducing ArcGIS Engine'.

ArcGIS Engine Runtime with Network extension can:

- Create, update, and delete networks on shapefiles, SDC, and pGDB.
- Solve any network.

ArcGIS Engine Runtime with the Geodatabase Update and the Network extension can:

- Create, update, and delete networks on shapefiles, SDC, pGDB, and GDB.
- Solve any network.

All editing of datasets is subject to the standard editing restrictions in the core ESRI products.

ARCGIS ENGINE AND ARCGIS DESKTOP LICENSING

One advantage of ArcGIS Engine mentioned in previous chapters is the fact that it is the platform used for the ArcGIS Desktop applications. This allows for more flexibility in the licensing of your application. In some cases, you may wish to deploy an ArcGIS Engine-based application that can utilize either the core ArcGIS Engine Runtime product and extension licenses or the equivalent ArcGIS Desktop product and extension licenses.

Methods for leveraging multiple levels of licensing in this manner are discussed in the section, 'License initialization'.

The following table lists the ArcGIS Engine product or extension license and its equivalent ArcGIS Desktop license.

ArcGIS Engine	ArcGIS Desktop	License comparison
ArcGIS Engine Runtime	ArcView	ArcGIS Engine Runtime is functionally equivalent to ArcView at the ArcObjects level. ArcGIS Engine Runtime does not include any of the ArcGIS Desktop applications, such as ArcMap, but it does include the same core GIS ArcObjects.
ArcGIS Engine with Geodatabase Update extension	ArcEditor	ArcGIS Engine Runtime with the Geodatabase Update extension is functionally equivalent to ArcEditor at the ArcObjects level. The Geodatabase Update extension does not include any of the advanced editing tools available in ArcMap, but it does provide complete access to the Geodatabase API for creating, editing, and managing an enterprise geodatabase.
ArcGIS Engine with 3D extension	ArcGIS Desktop (ArcView, ArcEditor, or ArcInfo) with 3D Analyst	ArcGIS Engine Runtime with the 3D extension is functionally equivalent to ArcGIS Desktop with the 3D Analyst extension at the ArcObjects level. The 3D extension for ArcGIS Engine does not include the ArcScene or ArcGlobe applications, but it does provide access to the GlobeControl and SceneControl for embedding 3D visualization in custom applications.
ArcGIS Engine with Spatial extension	ArcGIS Desktop (ArcView, ArcEditor, or ArcInfo) with Spatial Analyst	ArcGIS Engine Runtime with the Spatial extension is functionally equivalent to ArcGIS Desktop with the Spatial Analyst extension at the ArcObjects level. The Spatial extension for ArcGIS Engine does not include the ArcGIS Desktop toolbars or commands, but it does provide the ArcObjects components necessary for raster cell analysis.
ArcGIS Engine Runtime with the Network extension	ArcGIS Desktop (ArcView, ArcEditor, or ArcInfo) with Network Analyst	ArcGIS Engine Runtime with the Network extension is functionally equivalent to ArcGIS Desktop with the Network Analyst extension at the ArcObjects level. The ArcGIS Engine Network extension does not include the ArcGIS Desktop toolbars or commands, but it does provide the ArcObjects components necessary for creating custom applications that solve on any network.

The ArcGIS Engine Developer Kit is a software developer product containing all the developer components and resources required for building and deploying custom standalone desktop GIS and mapping applications. The ArcGIS Engine Developer Kit contains the following CDs:

- ArcGIS Engine Developer Kit: ArcGIS Engine and software developer kit for the COM, .NET, Java, and C++ APIs along with supporting developer resources. There is a separate CD for Windows, Red Hat and SUSE Linux, and Sun Solaris.
- ArcGIS Engine Runtime: Redistributable version of ArcObjects needed to run applications developed with the ArcGIS Engine Developer Kit. There is a separate CD for Windows, Red Hat and SUSE Linux, and Sun Solaris.
- ESRI Software Documentation Library: Digital versions (PDF) of all ArcGIS user guides associated with the ArcGIS system.
- ESRI Data and Maps: Media kit containing many types of map data at any scales of geography.

The ArcGIS Engine Developer Kit ships with registration numbers for building custom solutions and one copy of the ArcGIS Engine Runtime (including extensions) for testing purposes. You should receive an ECP registration number with the product packaging or via e-mail for the following:

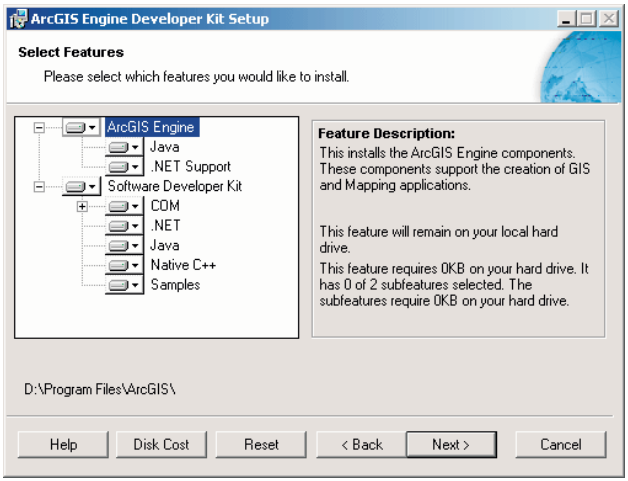
- ArcGIS Engine Developer Kit (includes the design-time ArcGIS Engine and all extensions)
- ArcGIS Engine Runtime for testing
- ArcGIS Engine Runtime Geodatabase Update extension for testing
- ArcGIS Engine Runtime 3D extension for testing
- ArcGIS Engine Runtime Spatial extension for testing
- ArcGIS Engine Runtime Network extension for testing

INSTALLATION OF ARCGIS ENGINE DEVELOPER KIT

To install it, insert the ArcGIS Engine Developer Kit CD into the CD drive to automatically start the setup program and follow the setup instructions. If any other ArcGIS products are installed on the machine, the install will default to the same location. ArcGIS COM libraries will be installed automatically by the install. If your chosen API is Java or .NET, you will need to install the Java or .NET ArcGIS Engine support features.

- The .NET feature will install a .NET assembly for each COM library. Prior to installing the .NET feature, Microsoft .NET Framework 1.1 must be installed on the machine.
- The Java feature will install a .jar file for each COM library. Prior to installing the Java feature, JDK 1.4.2 must be installed on the machine.

A separate developer kit can be installed for the COM, .NET, Java, and C++ APIs containing help, object model diagrams, tools, and any specific API components. Samples are provided for the ArcGIS Desktop, ArcGIS Engine, and ArcGIS Server products.



At the end of the installation of the ArcGIS Engine Developer Kit, the Software Authorization wizard appears, enabling you to either:

- Register the software with ESRI using the ArcGIS Engine Developer Kit registration code previously supplied, if you have not already done so, to receive an ESRI Copy Protection (*.ecp) file.
- Finish the registration process by authorizing the software with the .ecp file to unlock the underlying software components for development use.

REGISTRATION OF ARCGIS ENGINE DEVELOPER KIT

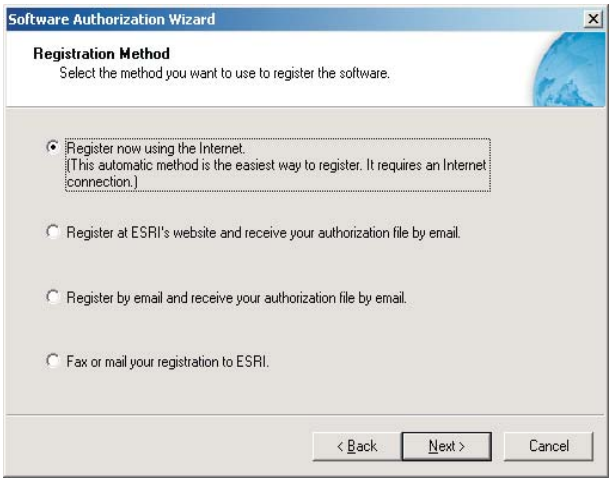
The ArcGIS Engine Developer Kit and the ArcGIS Engine Runtime need to be registered with ESRI and authorized for use on a per-machine, single-use basis via an .ecp file to unlock the underlying software components for use. This registration is a two-step process:

The registration codes supplied with the product packaging for ArcGIS Engine Runtime testing can be registered at the same time the ArcGIS Engine Developer Kit is registered.

Copy protection for ArcGIS Engine Developer Kit

To receive an ESRI Copy Protection file for the ArcGIS Engine Developer Kit, the registration code supplied with the product packaging must be registered with ESRI. The products can be registered either during installation or after installation via:

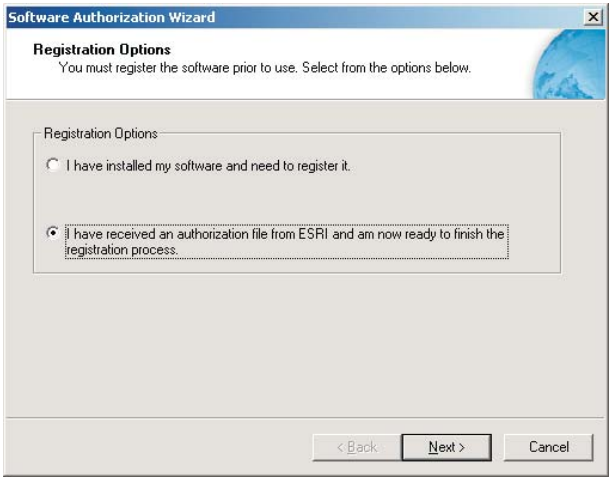
- The ESRI Customer Service Web site at *http://service.esri.com*.
- The Software Authorization Wizard at <install_location>\bin\SoftwareAuthorization.exe.



Software authorization for ArcGIS Engine Developer Kit

To complete the registration process the ArcGIS Engine Developer Kit must be authorized with the .ecp file to unlock the underlying software components for development use. The product can be authorized by:

- Using the Software Authorization Wizard found in <install_location>\bin\SoftwareAuthorization.exe to navigate to the .ecp file.
- Registering the ArcGIS Engine Developer Kit online at *http://service.esri.com* gives you the option to dynamically authorize the product over the Internet. In this case, no .ecp file will be sent.



- The Authorization Summary tool in <install_location>\bin\AuthorizationSummary.exe can be used to verify the license configuration of the ArcGIS Engine Developer Kit. All levels of functionality are available for you to develop with including the extension functionality.

The licensing and deployment models for custom applications must be considered before application development begins, alongside the functional requirements of the application, because a number of end user software and license configurations are possible.

LICENSING CONCEPTS

Each of the sample applications that you build in Chapter 6,'Developer scenarios', illustrates the license initialization process.

After you've determined which licensing options your application requires and which ones it will be able to run against, you must make sure that your application initializes those licenses correctly. This section describes the license initialization process, illustrates how a multiple license initialization would work, and provides a number of sample initialization processes.

Key concepts

- License initialization must be performed by the application, at application start time, before any ArcObjects components are accessed.
- Once an application has been initialized with a license, it cannot be reinitialized; an application is initialized with a license for the duration of its execution.
- The extension licenses available to an application come from the same license server as the license used to initialize the application; extension licenses on different license servers are not available to the application.

A license server is defined as the machine providing the license.

- Extension licenses can be either checked out for the duration of the application's life or can be checked out and checked in when required by the application.
- Always attempt to consume the lowest level license.

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 DLL that 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.

Each standalone application developed using ArcObjects must initialize itself with a suitable ArcGIS license to ensure it runs successfully on any machine to which it is deployed. The license initialization must be performed by the application, at application start time, before any ArcObjects components are accessed. Failure to initialize results in application errors.

As discussed in the previous section, 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 is made available in certain license flavors—Engine Single Use, Desktop Concurrent Use, and Desktop Single Use.

- Engine Single Use—Provides access to ArcGIS Engine and its extension 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 from 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, you would treat them the same.

Once an application has been initialized with a license, it cannot be reinitialized; an application is initialized with a license for the duration of its life. For example, you can't write an application that starts up with an ArcView license and later switches to ArcEditor.

When initializing an application with a license, the following must be considered:

As noted in the previous section, using an ArcView license with an ArcGIS Engine application will give you access to all the functionality available to a standard ArcGIS Engine license. Likewise, using an ArcEditor license with an ArcGIS Engine application that performs multiuser geodatabase editing will give you access to all the functionality available to an ArcGIS Engine license with the Geodatabase Update extension.

- The types of product licenses with which the application can run. 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 a Geodatabase Update extension license, an ArcEditor license, or an ArcInfo license.
- The types of product licenses 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.

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 different 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 it is recommended you use it instead of the Desktop Concurrent and Desktop Single Use licenses. This means you will not limit the Desktop Concurrent licenses available to any other users.

THE INITIALIZATION PROCESS

Applications built with any of the ArcGIS controls must also adhere to this license initialization process. MapControl, PageLayoutControl, ReaderControl, TOCControl, and ToolbarControl applications require the ArcGIS Engine Runtime, ArcView, ArcEditor, or ArcInfo product licenses. Since GlobeControl and SceneControl extend core ArcGIS Engine functionality, they require a corresponding 3D extension license in addition to the core product license.

The initialization of an application with a license must be performed in the following order:

1. Check whether the product license is available.
2. Check whether extension licenses are available (if required).
3. Initialize the application with the product license.

4. As required, perform extension checkouts and check-ins.
5. Shut down the application.

Step 1: Check 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 application's 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.

Step 2: Check 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 dependent on the extension.

Step 3: Initialize 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 reinitialize the application.

Step 4: Check extensions in and out

Extensions can either be checked out when an application requires the extension functionality and checked in once the application has finished with the functionality or 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 with which the application was initialized.

- 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 the extension functionality is required. When the extension is checked in, the functionality should be disabled.

Step 5: 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.

WHY DOES INITIALIZATION FAIL?

If a product or extension 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 its life. It is possible to check with which product license an application has been initialized. For example, if an application containing some enterprise geodatabase editing has been initialized with an Engine Single Use with Geodatabase Update extension 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.

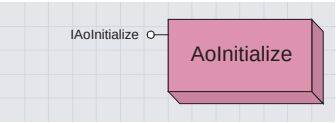
LICENSE CONFIGURATION

There are two ways to configure a custom application with a license: using the LicenseControl and programmatically.

Use the LicenseControl to automatically perform license initialization within simple graphical user interface applications using the MapControl, PageLayoutControl, TOCControl, ToolbarControl, ReaderControl, SceneControl, or GlobeControl in the COM and .NET APIs. If greater control is required over license initialization, particularly when checking extension licenses in and out (the LicenseControl will check out extension licenses for the duration of an application's life), consider programmatically performing license initialization.

For a detailed description on how to use the LicenseControl, see Chapter 3, 'Developing with ArcGIS controls', and Chapter 6, 'Developer scenarios'.

To programmatically perform license initialization use the *AoInitialize* object and the *IAoInitialize* and *ILicenseInformation* interfaces it implements. The ESRI License Initializer Visual Basic 6 and Visual Studio .NET add-ins can be used to automatically generate and add license initialization code to a custom application. See Appendix B, 'ArcGIS developer resources', for further information about the add-ins or the licensing samples found in <install_location>\DeveloperKit\Samples\Licensing_and_Extension_Checking.



The coclass AoInitialize *and the* IAoInitialize *and* ILicenseInformation *interfaces it implements are designed to give support to a developer for license initialization.*

EXAMPLE A—MINIMUM LICENSE IS ARCGIS ENGINE WITH 3D AND SPATIAL EXTENSIONS

In this example, the application requires at minimum an ArcGIS Engine license. In addition, this developer has decided that if an ArcGIS Engine license is not available, the application could run with an ArcView or ArcEditor license instead. The application also requires 3D and spatial extension functionality for it to run successfully, so both of these extensions need to be checked out for the duration of the application.

In this example, even though ArcInfo would also provide the functionality required, the developer has opted not to initialize with an ArcInfo license. It would be unnecessary to consume an ArcInfo license for this simple application.

In this case, the application will first attempt to initialize against the ArcGIS Engine product license. If that fails, it will attempt to initialize against an ArcView license, and if still unsuccessful, the application will finally attempt to initialize against the ArcEditor product license. The following sections outline the steps that must be taken to initialize the application with a license.

Attempting initialization with the ArcGIS Engine product license

As noted above, to run successfully the application requires at minimum an ArcGIS Engine product license along with the corresponding 3D and Spatial extension licenses. The application's first attempt at initialization should be against this minimal level of product licensing. The application's attempts at initialization follow the process discussed earlier.

Step 1: Check product license availability

1. Check whether an ArcGIS Engine product license is available. If yes, proceed to the next step in the initialization process. If not, discontinue this attempt and restart initialization with any secondary level of allowable product licensing.

Step 2: Check extension license availability

2. Determine whether a 3D extension license is available for the ArcGIS Engine product license.

If yes, proceed with checks for any other needed extensions. If not, discontinue this attempt and restart initialization with any secondary level of allowable product licensing.

3. Since Spatial extension functionality is also required for this application, check whether a Spatial extension license is available for the ArcGIS Engine product license.

   If yes, proceed with checks for any other needed extensions. If not, discontinue this attempt and restart initialization with any secondary level of allowable product licensing.

4. In this case, no other extension licenses are needed. Proceed to the next step in the initialization process.

Step 3: Initialize the application

5. Check out the ArcGIS Engine product license by initializing the application. If the license checked out, proceed to the next step in the initialization process. If the license failed to check out, discontinue this attempt and restart initialization with any secondary level of allowable product licensing.

Step 4: Check extensions in and out

6. Check out the 3D extension for the ArcGIS Engine product license. If the license checked out, proceed with checkout for any other needed extensions. If the license failed to check out, discontinue this attempt and restart initialization with any secondary level of allowable product licensing.

7. Check out the Spatial extension for the ArcGIS Engine product license. If the license checked out, proceed with checkout for any other needed extensions.

   If the license failed to check out, discontinue this attempt and restart initialization with any secondary level of allowable product licensing.

8. In this case, no other extension licenses are needed. If the extension licenses are checked out, the application has been successfully configured with licenses.

Step 5: Shutdown

9. The final step in the initialization process is ensuring that the licenses are released when the application is shut down.

Attempting initialization with the ArcView product license

In this example, a secondary level of licensing is available if the first ArcGIS Engine product level fails to initialize correctly. The application once again attempts to initialize by following the defined process.

Step 1: Check product license availability

1. Check whether an ArcView product license is available. If yes, proceed to the next step in the initialization process. If not, discontinue this attempt and restart initialization with any tertiary level of allowable product licensing.

Step 2: Check extension license availability

2. Determine whether a 3D Analyst extension license is available with the ArcView product license. If yes, proceed with checks for any other needed extensions. If not, discontinue this attempt and restart initialization with any tertiary level of allowable product licensing.

3. Check whether an ArcGIS Spatial Analyst extension license is available with the ArcView product license. If yes, since no other extensions are needed, proceed to the next step in the initialization process. If not, discontinue this attempt and restart initialization with any tertiary level of allowable product licensing.

Step 3: Initialize the application

4. Check out the ArcView product license by initializing the application. If the license checked out, proceed to the next step in the initialization process. If not, discontinue this attempt and restart initialization with any tertiary level of allowable product licensing.

Step 4: Check extensions in and out

5. Check out the 3D Analyst extension. If the license checked out, proceed with checkout for any other needed extensions. If the license failed to check out, discontinue this attempt and restart initialization with any tertiary level of allowable product licensing.

The checkout of licenses may fail even though the availability check was successful. This is particularly possible in cases in which Desktop Concurrent licenses were initially available but may have since been checked out by another application.

6. Check out the ArcGIS Spatial Analyst extension. Since no other extension licenses are needed, if the licenses are checked out, the application has been successfully configured with licenses. If not, discontinue this attempt and restart initialization with any tertiary level of allowable product licensing.

Step 5: Shutdown

7. The final step in the initialization process is ensuring that the licenses are released when the application is shut down.

Attempting initialization with the ArcEditor product license

In this example, if both the first (ArcGIS Engine) and second (ArcView) product levels fail to initialize correctly, a third level of licensing—ArcEditor—is available. The application makes a final attempt to initialize by following the defined process.

Step 1: Check product license availability

1. Check whether an ArcEditor product license is available. If yes, proceed to the next step in the initialization process. If not, discontinue this final attempt. The application cannot run successfully at this time.

Step 2: Check extension license availability

2. Determine whether a 3D Analyst extension license is available with the ArcEditor product license. If yes, proceed with checks for any other needed extensions. If not, discontinue this final attempt. The application cannot run successfully at this time.

3. Check whether an ArcGIS Spatial Analyst extension license is available with the ArcEditor product license. If yes, since no other extensions are needed, proceed to the next step in the initialization process. If not, discontinue this final attempt. The application cannot run successfully at this time.

Step 3: Initialize the application

4. Check out the ArcEditor product license by initializing the application. If the license checked out, proceed to the next step in the initialization process. If the license failed to check out, discontinue this final attempt. The application cannot run successfully at this time.

Step 4: Check extensions in and out

5. Check out the 3D Analyst extension. If the license checked out, proceed with checkout for any other needed extensions. If the license failed to check out, discontinue this final attempt. The application cannot run successfully at this time.

The checkout of licenses may fail even though the availability check earlier in the process was successful.This is particularly possible in cases where Desktop Concurrent licenses—instead of Single Use ones—were initially available but may have since been checked out by another application.

6. Check out the ArcGIS Spatial Analyst extension. Since no other extension licenses are needed, if the licenses are checked out, the application has been successfully configured with licenses. If not, discontinue this final attempt. The application cannot run successfully at this time.

Step 5: Shutdown

7. The final step in the initialization process is ensuring that the licenses are released when the application is shut down.

EXAMPLE B—MINIMUM LICENSE IS ARCGIS ENGINE WITH GEODATABASE UPDATE AND SPATIAL EXTENSIONS

In the next example, the application is an enterprise geodatabase editing application and, therefore, requires the minimum of an ArcGIS Engine with Geodatabase Update extension license. Additionally, this developer has decided that if an ArcGIS Engine with Geodatabase Update extension license is not available, the application could run with an ArcEditor or ArcInfo license instead. The application also requires spatial extension functionality for it to run successfully, so the extension needs to be checked out for the duration of the application. In this case, the application will first attempt to initialize against the ArcGIS Engine with Geodatabase Update extension product license. If that fails, it will attempt to initialize against an ArcEditor license, and if still unsuccessful, the application will finally attempt to initialize against the ArcInfo product license. The following sections outline the steps taken to initialize the application with a license.

Attempting initialization with the ArcGIS Engine with Geodatabase Update product license

As noted above, to run successfully the application requires at minimum an ArcGIS Engine with Geodatabase Update extension product license along with the corresponding Spatial extension license. The application's first attempt at initialization should be against this minimal level of product licensing.

The application's attempts at initialization follow the process discussed earlier:

Step 1: Check product license availability

1. Check whether an ArcGIS Engine with Geodatabase Update extension product license is available. If yes, proceed to the next step in the initialization process.

   If not, discontinue this attempt and restart initialization with any secondary level of allowable product licensing.

Step 2: Check extension license availability

2. Check whether an Spatial extension license is available with the ArcGIS Engine with Geodatabase Update extension product license. If yes, proceed to the next step in the initialization process. If not, discontinue this attempt and restart initialization with any secondary level of allowable product licensing.

Step 3: Initialize the application

3. Check out the ArcGIS Engine with Geodatabase Update extension product license by initializing the application. If the license checked out, proceed to the next step in the initialization process. If the license failed to check out, discontinue this attempt and restart initialization with any secondary level of allowable product licensing.

Step 4: Check extensions in and out

4. Check out the Spatial extension. Since no other extension licenses are needed, if the license checked out, the application has been successfully configured with licenses.

   If the license failed to check out, discontinue this attempt and restart initialization with any secondary level of allowable product licensing.

Step 5: Shutdown

5. The final step in the initialization process is ensuring that the licenses are released when the application is shut down.

Attempting initialization with the ArcEditor product license

In this example, a secondary level of licensing is available if the first ArcGIS Engine product level fails to initialize correctly. The application once again attempts to initialize by following the defined process.

Step 1: Check product license availability

1. Check whether an ArcEditor product license is available. If yes, proceed to the next step in the initialization process. If not, discontinue this attempt and restart initialization with any tertiary level of allowable product licensing.

Step 2: Check extension license availability

2. Check whether an ArcGIS Spatial Analyst extension license is available with the ArcEditor product license. If yes, proceed to the next step in the initialization process. If not, discontinue this attempt and restart initialization with any tertiary level of allowable product licensing.

Step 3: Initialize the application

The checkout of licenses may fail even though the availability check was successful.This is particularly possible in cases where Desktop Concurrent licenses were initially available but may have since been checked out by another application.

3. Check out the ArcEditor product license by initializing the application. If the license checked out, proceed to the next step in the initialization process.

   If the license failed to check out, discontinue this attempt and restart initialization with any tertiary level of allowable product licensing.

Step 4: Check extensions in and out

4. Check out the ArcGIS Spatial Analyst extension. Since no other extension licenses are needed, if the license checked out, the application has been successfully configured with licenses.

   If the license failed to check out, discontinue this attempt and restart initialization with any tertiary level of allowable product licensing.

Step 5: Shutdown

5. The final step in the initialization process is ensuring that the licenses are released when the application is shut down.

Attempting initialization with the ArcInfo product license

In this example, if both the first (ArcGIS Engine) and second (ArcEditor) product levels fail to initialize correctly, a third level of licensing is available, ArcInfo. The application makes a final attempt to initialize by following the defined process.

Step 1: Check product license availability

1. Check whether an ArcInfo product license is available. If yes, proceed to the next step in the initialization process. If not, discontinue this final attempt. The application cannot run successfully at this time.

Step 2: Check extension license availability

2. Check whether an ArcGIS Spatial Analyst extension license is available with the ArcInfo product license. If yes, proceed to the next step in the initialization process. If not, discontinue this final attempt. The application cannot run successfully at this time.

Step 3: Initialize the application

The checkout of licenses may fail even though the availability check earlier in the process was successful. This is particularly possible in cases in which Desktop Concurrent licenses—instead of Single Use ones—were initially available but may have since been checked out by another application.

3. Check out the ArcInfo product license by initializing the application. If the license failed to check out, discontinue this final attempt. The application cannot run successfully at this time.

Step 4: Check extensions in and out

4. Check out the ArcGIS Spatial Analyst extension. Since no other extension licenses are needed, if the licenses are checked out, the application has been successfully configured with licenses. If the license failed to check out, discontinue this final attempt. The application cannot run successfully at this time.

Step 5: Shutdown

5. The final step in the initialization process is ensuring that the licenses are released when the application is shut down.

EXAMPLE C—MINIMUM LICENSE IS ARCGIS ENGINE. OPTIONAL NETWORK FUNCTIONALITY AVAILABLE

Once again the example application requires a minimum of an ArcGIS Engine license. The developer has decided that if an ArcGIS Engine license is not available, the application could run with an ArcView, ArcEditor, or ArcInfo license instead. Similar to the previous examples, this application also includes extension functionality—Network—but, in this case, the extension is not required simply to run the application; instead, it enables additional functionality. As such the Network extension will be checked out dynamically during the use of the application rather than at application startup.

Once again the application will first attempt to initialize against the ArcGIS Engine product license. If that fails, it will make additional attempts to initialize against an ArcView license, an ArcEditor license, and if still unsuccessful, the application will finally attempt to initialize against the ArcInfo product license. The following sections outline the steps taken to initialize the application with a license.

Attempting initialization with the ArcGIS Engine product license

As indicated above, to run successfully the application requires at minimum an ArcGIS Engine product license. While Network functionality is available within the application, its extension license can be checked out when needed rather than for the duration of the session. The application's first attempt at initialization should be against this minimal level of product licensing. The application's attempts at initialization follow the process discussed earlier.

Step 1: Check product license availability

1. Check whether an ArcGIS Engine product license is available. If yes, proceed to the next step in the initialization process. If not, discontinue this attempt and restart initialization with any secondary level of allowable product licensing.

Step 2: Check extension license availability

2. Check whether a Network extension license is available with the ArcGIS Engine product license. If yes, proceed to the next step in the initialization process. If not, proceed to the next step in the initialization process.

Step 3: Initialize the application

3. Check out the ArcGIS Engine product license by initializing the application. If the license checked out, proceed to the next step in the initialization process. If the license failed to check out, discontinue this attempt and restart initialization with any secondary level of allowable product licensing.

   Since this application dynamically checks out the Network extension, this step is not performed at this time. Instead, the license will be checked out during usage of Network functionality. See the section 'Using Network functionality' below for details on this process.

Step 4: Check extensions in and out

   In this example, no licenses beyond the ArcGIS Engine product license need to be checked out, so if that license checked out, the application has been successfully configured with licenses.

Step 5: Shutdown

4. The final step in the initialization process is ensuring that the licenses are released when the application is shut down.

Attempting initialization with the ArcView product license

In this example, a secondary level of licensing is available if the first ArcGIS Engine product level fails to initialize correctly. The application once again attempts to initialize by following the defined process:

Step 1: Check product license availability

1. Check whether an ArcView product license is available. If yes, proceed to the next step in the initialization process. If not, discontinue this attempt and restart initialization with any tertiary level of allowable product licensing.

Step 2: Check extension license availability

2. Check whether an ArcGIS Network extension license is available with the ArcView product license. If yes, proceed to the next step in the initialization process. If not, discontinue this attempt and restart initialization with any tertiary level of allowable product licensing.

Step 3: Initialize the application

3. Check out the ArcView product license by initializing the application. If the license checked out, proceed to the next step in the initialization process. If the license failed to check out, discontinue this attempt and restart initialization with any tertiary level of allowable product licensing.

   Step 4: Check extensions in and out

   Since this application dynamically checks out the ArcGIS Network extension, this step is not performed at this time. Instead, the license will be checked out during usage of Network functionality. See the section 'Using Network functionality' below for details on this process.

   In this example, no licenses beyond the ArcGIS Engine product license need to be checked out, so if that license checked out, the application has been successfully configured with licenses.

Step 5: Shutdown

4. The final step in the initialization process is ensuring that the licenses are released when the application is shut down.

Attempting initialization with the ArcEditor product license

In this example, if both the first (ArcGIS Engine) and second (ArcView) product levels fail to initialize correctly, a third level of licensing—ArcEditor—is available. The application makes another attempt to initialize by following the defined process:

Step 1: Check product license availability

1. Check whether an ArcEditor product license is available. If yes, proceed to the next step in the initialization process. If not, discontinue this attempt and restart initialization with any fourth level of allowable product licensing.

Step 2: Check extension license availability

2. Check whether an ArcGIS Network extension license is available with the ArcEditor product license. If yes, proceed to the next step in the initialization process. If not, discontinue this attempt and restart initialization with any fourth level of allowable product licensing.

Step 3: Initialize the application

3. Check out the ArcEditor product license by initializing the application. If the license checked out, proceed to the next step in the initialization process. If the license failed to check out, discontinue this attempt and restart initialization with any fourth level of allowable product licensing.

   Since this application dynamically checks out the ArcGIS Network extension, this step is not performed at this time. Instead, the license will be checked out during usage of Network functionality. See the section 'Using Network functionality' below for details on this process.

Step 4: Check extensions in and out

In this example, no licenses beyond the ArcGIS Engine product license need to be checked out, so if that license checked out, the application has been successfully configured with licenses.

Step 5: Shutdown

4. The final step in the initialization process is ensuring that the licenses are released when the application is shut down.

Attempting initialization with the ArcInfo product license

If the first (ArcGIS Engine), second (ArcView), and third (ArcEditor) product levels fail to initialize correctly, one final level of licensing—ArcInfo—is available.

The application makes a final attempt to initialize by following the defined process.

Step 1: Check product license availability

1. Check whether an ArcInfo product license is available. If yes, proceed to the next step in the initialization process. If not, discontinue this attempt and restart initialization with any fourth level of allowable product licensing.

Step 2: Check extension license availability

2. Check whether an ArcGIS Network extension license is available with the ArcInfo product license. If yes, proceed to the next step in the initialization process. If not, discontinue this attempt and restart initialization with any fourth level of allowable product licensing.

Step 3: Initialize the application

3. Check out the ArcInfo product license by initializing the application. If the license checked out, proceed to the next step in the initialization process. If the license failed to check out, discontinue this attempt and restart initialization with any fourth level of allowable product licensing.

   Since this application dynamically checks out the ArcGIS Network extension, this step is not performed at this time. Instead, the license will be checked out during usage of Network functionality. See the section 'Using Network functionality' below for details on this process.

Step 4: Check extensions in and out

In this example, no licenses beyond the ArcGIS Engine product license need to be checked out, so if that license checked out, the application has been successfully configured with licenses.

Step 5: Shutdown

4. The final step in the initialization process is ensuring that the licenses are released when the application is shut down.

Using Network functionality

During each of the initialization attempts in this example, the checkout of the Network extension did not occur during application startup. Instead, the checkout occurs dynamically when the Network functions are accessed within the application. This means that the Network license continues to be available to other users when not in use by this application.

As discussed earlier in the chapter, the extension license must be of the same product type as the base license. If the application initialized with an ArcGIS Engine product license, then the Network functionality cannot be initialized unless an ArcGIS Engine extension license for Network is available. When the application user attempts to perform any Network functions, the application performs the following steps to activate the needed functions:

1. Check whether the Network extension is already checked out. If the license is checked out, the application can use the Network functionality. If the license is not checked out, check the license out.
2. Check out the Network extension. If the license failed to check out, then the application cannot use the Network functionality. If the license is checked out, the application can use the Network extension.

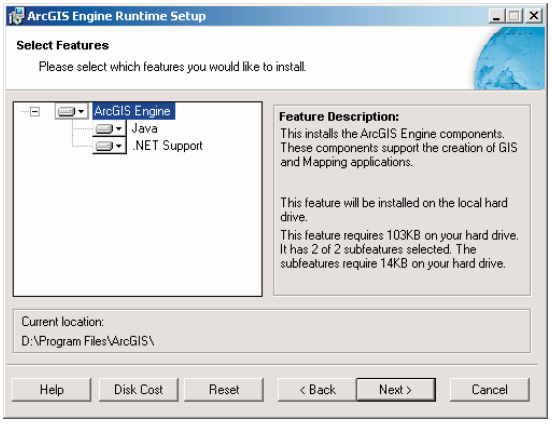
The checkout of licenses may fail even though the availability check was successful.This is particularly possible in cases where Desktop Concurrent licenses were initially available but may have since been checked out by another application.

After developing a custom standalone application, it should be tested with ArcGIS Engine Runtime before it is deployed to end user machines. Supplied as part of the ArcGIS Engine Developer Kit is a redistributable version of the ArcGIS Engine Runtime and registration numbers for the runtime testing of custom applications.

INSTALLATION OF ARCGIS ENGINE RUNTIME FOR TESTING

To install it, insert the ArcGIS Engine Runtime CD in the CD drive and use Windows Explorer to navigate to the CD drive. Follow the instructions of the ArcGIS Engine Runtime Setup. If any other ArcGIS products are installed on the machine, the install will default to the same location. ArcGIS COM libraries will be installed automatically by the install. If your custom application was developed with the .NET or Java APIs, you will need to install the Java or .NET ArcGIS Engine support features.

- The .NET feature will install a .NET assembly for each COM library. Prior to installing the .NET feature, Microsoft .NET Framework 1.1 must be installed on the machine.
- The Java feature will install a .jar file for each COM library. Prior to installing the Java feature, JDK 1.4.2 must be installed on the machine.



REGISTRATION OF ARCGIS ENGINE RUNTIME FOR TESTING

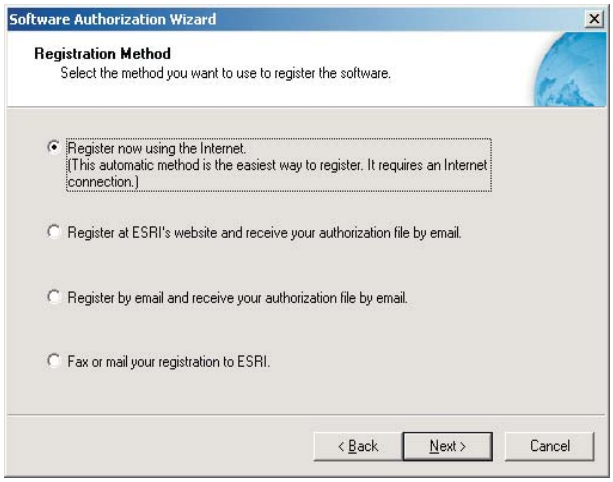
ArcGIS Engine Runtime, just like the ArcGIS Engine Developer Kit, needs to be registered with ESRI and authorized for use on a per-machine, single-use basis via a .ecp file to unlock the underlying software components for runtime use. This registration is a two-step process.

Copy protection for ArcGIS Runtime for testing

The registration codes supplied with the product packaging for ArcGIS Engine Runtime testing can be registered at the same time the ArcGIS Engine Developer Kit is registered.

To receive an ESRI Copy Protection file for ArcGIS Engine Runtime, one or more testing registration numbers supplied with the product packaging must be registered with ESRI. You may have already registered the testing registration codes when registering the ArcGIS Engine Developer Kit. The products can be registered either before installation or after installation via:

- The ESRI Customer Service Web site: *http://service.esri.com*.
- The Software Authorization Wizard at <install_location>\bin\SoftwareAuthorization.exe.

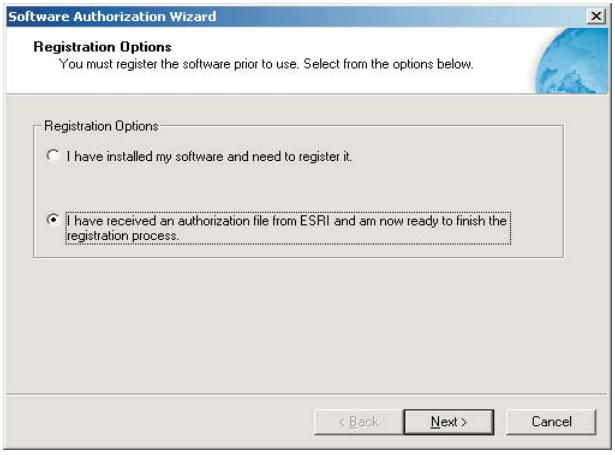


Software authorization for ArcGIS Runtime for testing

To complete the registration process, ArcGIS Engine Runtime must be authorized with the .ecp file to unlock the underlying software components for runtime use. 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 and prompted you to navigate to the .ecp file issued when you registered the product. Only after completing the software authorization were you able to develop and run custom ArcGIS Engine applications.

The product can be authorized by:

- Using the Software Authorization Wizard at <install_location>\bin to navigate to the .ecp file.
- Registering the ArcGIS Engine Runtime online at *http://service.esri.com* gives you the option to dynamically authorize the product over the Internet. In this case, no .ecp file will be sent.



The Authorization Summary tool at <install_location>\bin\AuthorizationSummary.exe can be used to verify the license configuration of the ArcGIS Engine Runtime and its extensions.

Once the functionality and the license initialization have been tested for the custom application, a setup may be created to deploy the application to end user machines. This setup may incorporate the custom application, map documents and data, additional components and resources, the ArcGIS Engine Runtime setup, and the automatic authorization of the ArcGIS Engine Runtime software. This is discussed in more details in the following 'Deployment' section. The setup itself should be tested to simulate the end user experience.

To successfully deploy custom applications to end user machines, ArcGIS Engine Runtime will need to be installed, the custom application will need to be installed, and ArcGIS Engine Runtime may need to be software authorized depending on the current machine license configurations.

DEPLOYING ARCGIS ENGINE RUNTIME

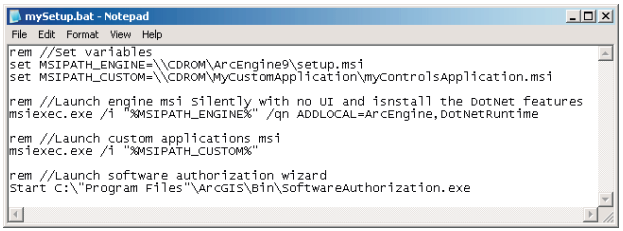
The ArcGIS Engine Runtime must be installed on every end user machine to which a custom application is deployed. This includes machines that may have ArcGIS Desktop installed, where the custom application will initialize itself with an ArcGIS Desktop license. As a purchaser of the ArcGIS Engine Developer Kit, you can freely distribute ArcGIS Engine Runtime, either on a CD or within your custom applications installation program. See Appendix D, 'Installing ArcGIS Engine Runtime on Windows, Solaris, and Linux', for further details on installing ArcGIS Engine Runtime.

DEPLOYING CUSTOM APPLICATIONS

The custom application is deployed to an end user machine that does not have ArcGIS Engine Runtime installed but has ArcGIS Desktop installed. In this case ArcGIS Engine Runtime must be installed on the machine, but the custom application initializes itself with the existing ArcGIS Desktop license.

The method used to create a setup for the custom application will have been decided on at the beginning of the development project and will depend on your organization, the end user requirements, and any previous experience you have of creating setups. The custom setup may use Microsoft Windows Installer (MSI) technology, may be a scripted setup or may simply be a batch file that is distributed on a CD to the end user.

The following is an example of a batch file that installs both the ArcGIS Engine Runtime and a custom application from setups on a CD, then launches the software authorization wizard for the end user.

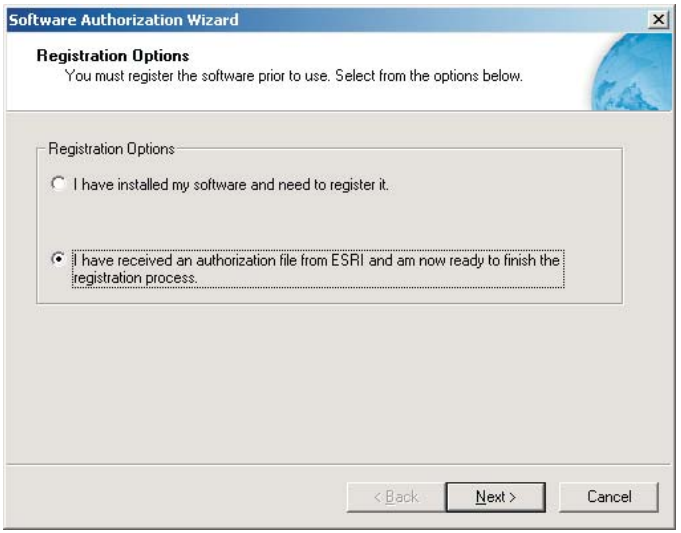


COPY PROTECTION AND SOFTWARE AUTHORIZATION FOR END USERS

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 '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 are a number of different ways to achieve authorization.



The Software Authorization Wizard opens after installing the 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, its authorizations, or both 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.

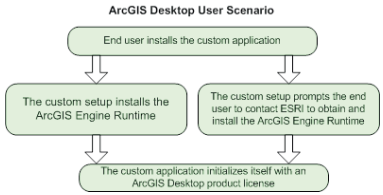
The following sections discuss the software authorization process for each of these three user types.

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 it has 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 , together with the ArcGIS Engine Runtime if its not already installed. See the previous 'Installation of ArcGIS Engine Runtime for testing' section for more details.
4. Upon application startup, it initializes and checks out an available license from the client's previously existing authorization or keycode file.

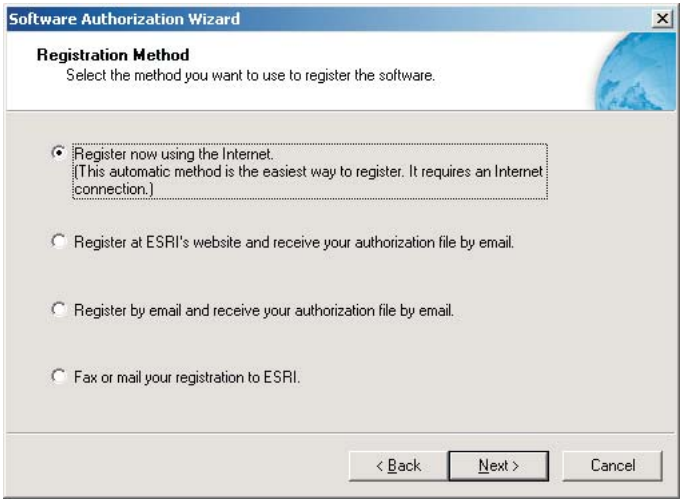


USER ACQUIRES ARCGIS ENGINE RUNTIME DIRECTLY FROM ESRI

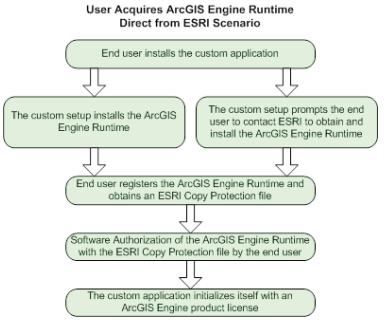
The second type of end user purchases, authorizes, or both the ArcGIS Engine Runtime software themselves. You and your client would go through the following process to install and run an application that you built:

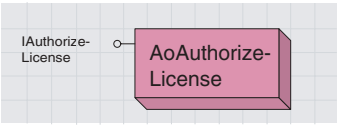
1. You review and confirm licensing required by your application.
2. Your client purchases ArcGIS Engine Runtime and any needed extensions (3D, Geodatabase, Spatial, Network, and so on), as determined in the previous step.
3. Your client registers the ArcGIS Engine product, and extensions if necessary, with ESRI (*http://www.service.esri.com*).
4. Your client receives an authorization file (.ecp) from ESRI and saves it to their computer.

If your clients don't register and receive their authorization file in advance, the Software Authorization Wizard directs them to do so.



5. Your client installs the ArcGIS Engine Runtime software.
6. Once the installation is complete, your client opens the Software Authorization Wizard.
   - On Windows, your client navigates to the \ArcGIS\bin folder and runs the SoftwareAuthorization.exe file it contains.
   - On Solaris and Linux, your client navigates to the \ArcGIS folder and runs the AuthorizeSoftware file 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 startup, it initializes and checks out an available license from the client's authorization file.





See ArcGIS Developer Help for more information on the IAuthorizeLicense *interface.*

The custom application is deployed to an end user machine. The custom setup itself installs and authorizes the ArcGIS Engine Runtime software so the end user has no contact with ESRI.

Use restrictions

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 'redistributed' 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.

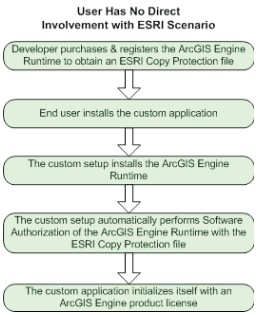
To run the SoftwareAuthorization tool from within your application's installation program, use the following argument:
/LIF <filename> /S

The /S triggers the tool to run silently with no user interface displaying.

USER HAS NO DIRECT INVOLVEMENT WITH ESRI

The final type of end user has no direct contact with ESRI. Instead your application calls the SoftwareAuthorization.exe file or the *IAuthorizeLicense* object, contained in your installation program or application, to unlock the functionality of ArcGIS Engine. This would require that you hard code the authorization keycode 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 extensions (3D, Geodatabase, Spatial, and so on), as determined in the previous step.
3. You register the ArcGIS Engine product, and extensions if necessary, with ESRI (*http://www.service.esri.com*).
4. You receive a redistributable authorization file (.ecp) and add its features to the code for your application.
5. Your client installs your custom-built ArcGIS Engine application. This:
   a. Installs ArcGIS Engine Runtime software
   b. Automatically runs the Software Authorization Wizard or uses the coclass *AoAuthorizeLicense* and the *IAuthorizeLicense* interface it implements
6. Upon application startup, 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, patterns, and 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 applications with C++ and Motif widgets • 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, click Custom or Modify, and click the samples feature under Software developer kit.

You can find this sample in:

```
<install_location>\DeveloperKit\samples\Developer_Guide_Scenarios\
Building_an_ArcGIS_Controls_Map_Viewer_ApplicationVisual_Basic.zip
```

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, ToolbarControl, and LicenseControl 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 preauthored ESRI map documents, or MXDs. The scenario covers the following techniques:

- Loading and embedding the ArcGIS controls in Microsoft Visual Basic 6.0
- Loading preauthored 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
- License configuration using the LicenseControl
- Deploying the application onto a Windows operating system

CONCEPTS

ActiveX is another term for a Microsoft Component Object Model 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 such as 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++ and Microsoft Visual Basic for Applications. 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, and the LicenseControl is used to configure the application with an appropriate license. 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 to highlight how the ArcGIS controls interact with each other and to expose a part of each ArcGIS control's 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, are advantageous.
- Access to the sample data and code that comes with this scenario. This is located at:

  `<install_location>\DeveloperKit\samples\Developer_Guide_Scenarios\Building_an_ArcGIS_Controls_Map_Viewer_ApplicationVisual_Basic.zip`

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, click Custom or Modify, and click the samples feature under Software Developer Kit.

The controls and libraries used in this scenario are as follows:

- LicenseControl
- MapControl
- PageLayoutControl
- TOCControl
- SystemUI Object Library
- Carto Object Library
- Display Object Library
- Geometry Object Library
- ToolbarControl

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 click Components.
3. In the Components dialog box, check ESRI MapControl, ESRI PageLayoutControl, ESRI TOCControl, ESRI ToolbarControl, and ESRI LicenseControl. 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 click Add-In Manager from the Add-Ins menu, click ESRI Automatic References, and check the load behavior check boxes. To then display the Add-In, click ESRI Automatic References from the Add-Ins menu.

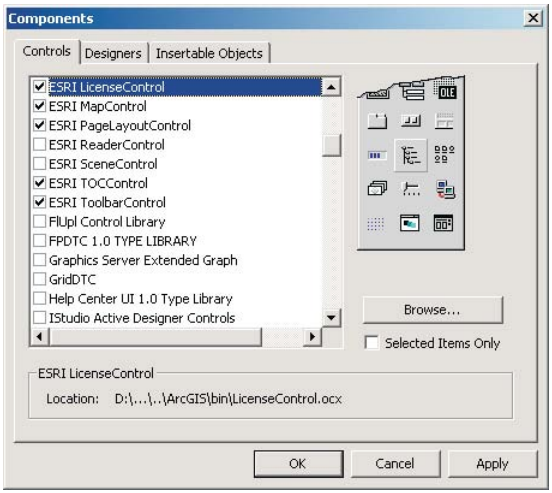

ESRIMapControl


ESRIPageLayoutControl


ESRITOCControl

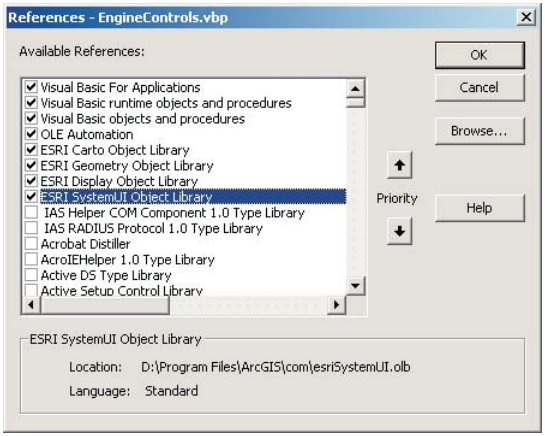

ESRIToolbarControl


ESRILicenseControl



The controls will now appear in the Visual Basic toolbox.

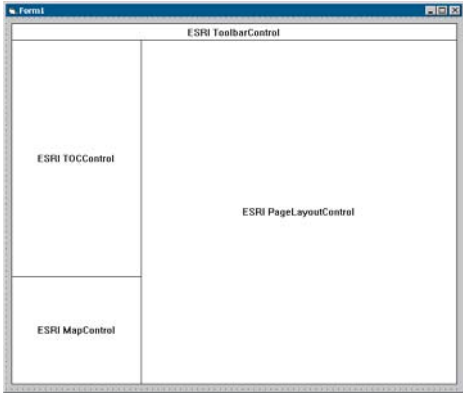
4. Click the Project menu again and click References.
5. In the References dialog box, check ESRI Carto Object Library, ESRI Display Object Library, ESRI Geometry Object Library, and ESRI SystemUI Object Library. Click OK.



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 button in the Visual Basic toolbox to add a MapControl to 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 preauthored ESRI map documents can be loaded into the MapControl and PageLayoutControl. You can either load the sample map document provided or you can load your own map document. Later you will add a dialog box to browse to a map document.

1. Double-click the form to display the code window.
2. Click 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 preauthored 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. Click 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 preauthored 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

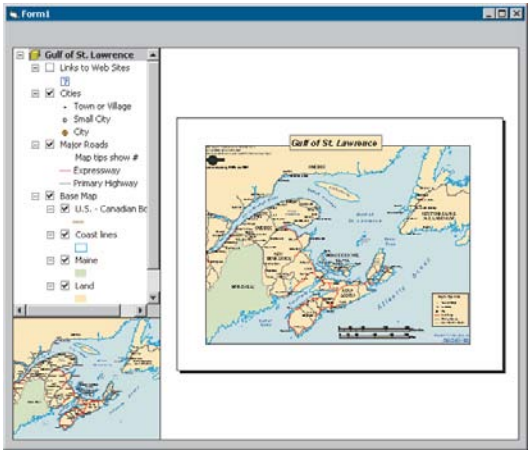
For the purpose of this application, the TOCControl and ToolbarControl will work in conjunction with the PageLayoutControl rather than the MapControl. To do this the PageLayoutControl must be set as the buddy control. The TOCControl uses the buddy's ActiveView to populate itself with maps, layers, and symbols, while any command, tool, or menu items present on the ToolbarControl will interact with the buddy control's display.

1. Double-click 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 preauthored 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 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 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 the form to display the code window.
2. Click 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. Click 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

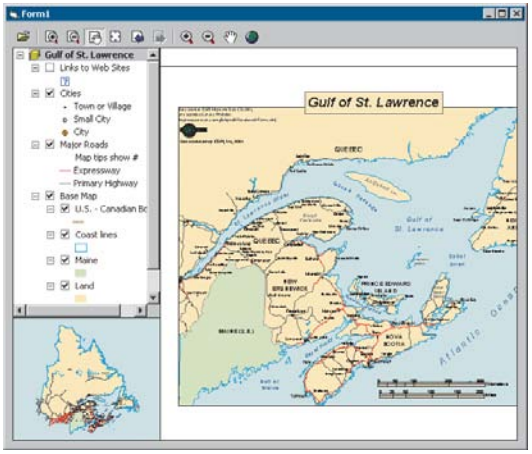
ArcGIS Engine comes with more than 120 commands and tools that work with the MapControl, the PageLayoutControl, and the ToolbarControl directly. These commands and tools provide you with a lot of frequently used GIS functionality for map navigation, graphics management, and feature selection. You will now add some of these commands and tools to your application.

1. Double-click the form to display the code window.
2. Click 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 navigation 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 preauthored…

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

In addition to 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 to your application that works with the PageLayoutControl. 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
```

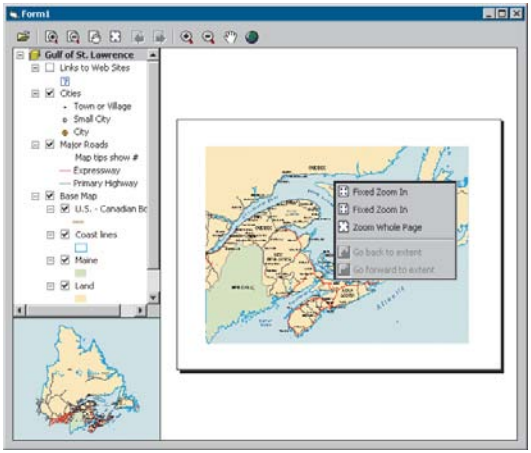
2. Add the following code to the Form_Load event after the code, adding the commands to the ToolbarControl but before the load document code.

```
Private Sub Form_Load()

  ' Add Map navigation 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 preauthored…

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 the PageLayoutControl 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.

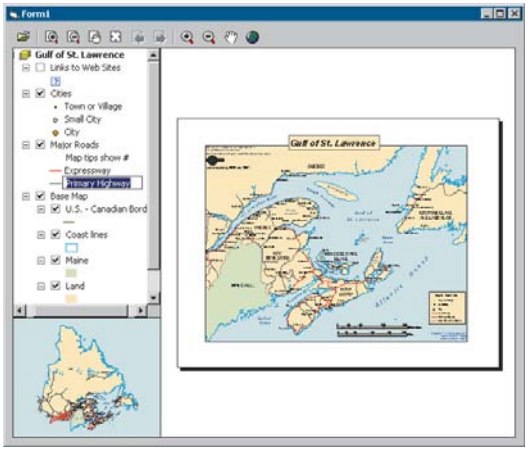
```
Private Sub Form_Load()

  ' Set label editing to manual.
  TOCControl1.LabelEdit = esriTOCControlManual

  ' Add generic commands…

End Sub
```

2. Add the following code to the TOCControl1_OnEndLabelEdit event.

```
Private Sub TOCControl1_OnEndLabelEdit(ByVal x As Long, ByVal y As Long,
  ByVal newLabel As String, CanEdit As Boolean)

  ' If the new label is an empty string, then prevent the edit.
  If Trim(newLabel) = "" Then CanEdit = False

End Sub
```

3. Run the application. To edit a map, layer, heading, or legend class label in the TOCControl, click it once, and click 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

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.*

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.

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 private subroutine 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 subroutine:

```
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* subroutine 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 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 ITransformEvents of the PageLayoutControl's focus map.
    Set m_pTransformEvents = pActiveView.ScreenDisplay.DisplayTransformation
    ' Get the extent of the focus map.
    Set m_pEnvelope = pActiveView.Extent

    ' Load the same preauthored 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
```

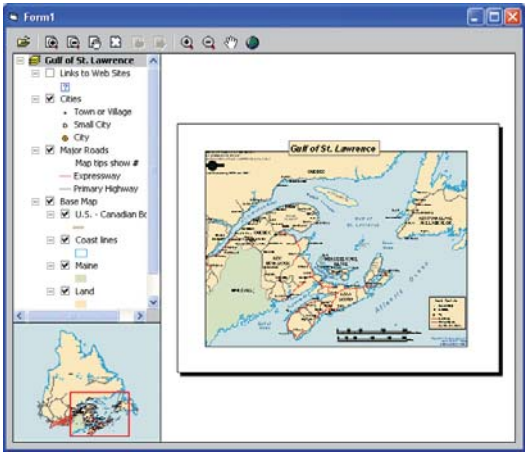
5. Add the following code to the m_pTransformEvents_VisibleBoundsUpdated event. This event is triggered whenever the extent of the map is changed and is used to set the envelope to the new visible bounds of the map. By refreshing the MapControl you force it to redraw the shape on its display.

```
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 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 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 click 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\Building_an_ArcGIS_Controls_Map_Viewer_ApplicationVisual_Basic.zip.

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
```

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:

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 a* MapControl, PageLayoutControl, *or* ToolbarControl*).*

```
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
```

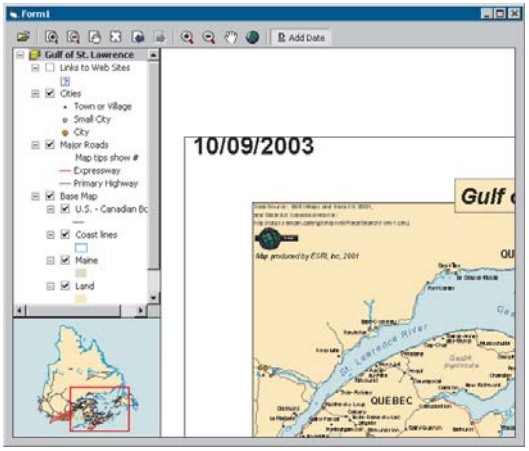
```
    ' Set the symbol properties.
    pTextSymbol.Font = pFont
    ' Set the text element properties.
    pTextElement.Symbol = pTextSymbol
    pTextElement.Text = Date

    ' 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

In addition to adding control 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 subroutine 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 subroutine:

```
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.
    m_pCustomizeDialog.SetDoubleClickDestination ToolbarControl1

End Sub
```

3. Call the *CreateCustomizeDialog* subroutine from the Form_Load event before the call to the *CreateOverviewSymbol* subroutine.

```
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
```

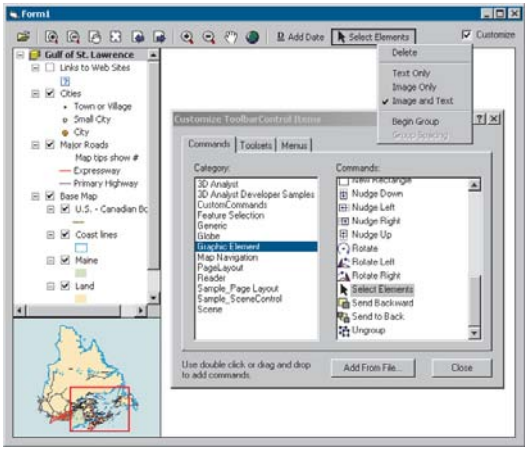
6. Add the following code to the m_pCustomizeDialogEvents_OnCloseDialog and m_pCustomizeDialogEvents_OnStartDialog events.

```
Private Sub m_pCustomizeDialogEvents_OnCloseDialog()

    ToolbarControl1.Customize = False
    chkCustomize.Value = 0

End Sub

Private Sub m_pCustomizeDialogEvents_OnStartDialog()

    ToolbarControl1.Customize = True

End Sub
```

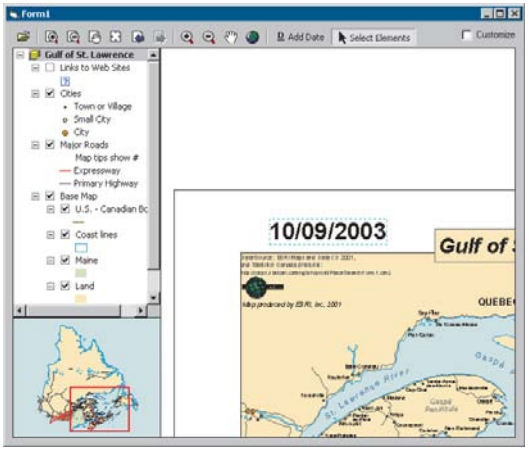
7. Run the application and check the Customize box to put the ToolbarControl into customize mode and open the Customize dialog box.
8. On the Commands tab select the Graphic Element category and double-click the Select Elements command to add it to the ToolbarControl. By right-

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.

clicking an item on the ToolbarControl, you can adjust its 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.



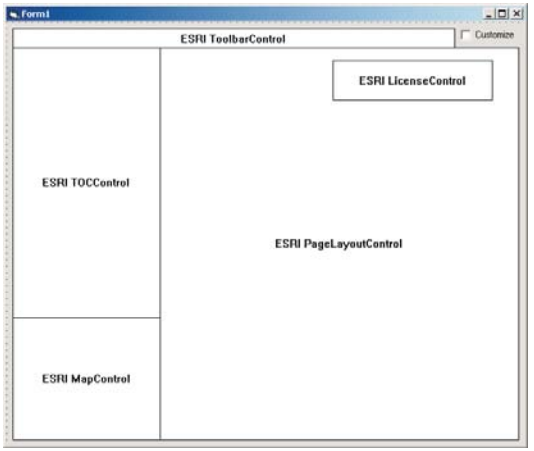
When developing a standalone executable using ESRI ArcObjects, it is the responsibility of the application to check and configure the licensing options. A license can be configured either using the LicenseControl or using the coclass AoInitialize *and the* IAoInitialize *interface it implements that are designed to support license configuration. License initialization must be performed at application start time, before any ArcObjects functionality is accessed. Failure to do so will result in application errors. For more information about licensing see the 'Licensing and deployment' chapter.*

The LicenseControl will appear on a form at design time so that it can be selected and its property pages viewed. However, at runtime the LicenseControl is invisible so its position on the form is irrelevant.

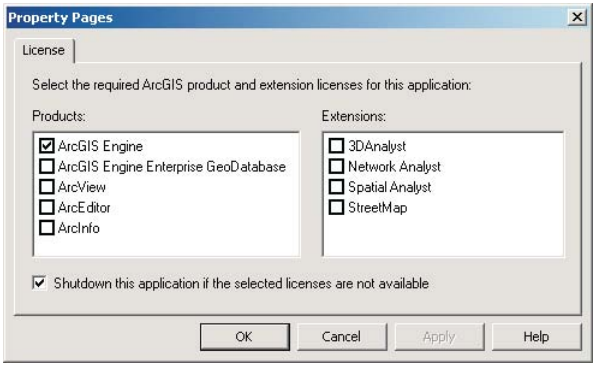
LICENSE CONFIGURATION AND DEPLOYMENT

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. License configuration can be performed either using the LicenseControl or programmatically using the *AoInitialize* object. For the purpose of this application, the LicenseControl will be used to handle license configuration.

1. Open the Visual Basic Form.
2. Double-click the LicenseControl button in the Visual Basic toolbox to add the LicenseControl to the form.



3. Right-click on LicenseControl and click Properties to open the LicenseControl property pages.
4. Check the ArcGIS Engine product license and check 'Shutdown this application if the selected licenses are not available'. Click OK.



5. Compile the application into an executable.

This application can be initialized with an ArcGIS Engine license, but you may optionally initialize the application with a higher product license. For example, if you check the 'ArcGIS Engine' license and the 'ArcView' license, the LicenseControl will initially try to initialize the application with an ArcGIS Engine license (the lower license). If that license is not available, the LicenseControl will try to initialize the application with an ArcView license (the next higher level license checked). If no product licenses are available, then the application will fail to initialize.

In this application the LicenseControl will handle license initialization failure. If the application cannot be initialized with an 'ArcGIS Engine' product license, a License Failure dialog box will be displayed to the user before the application is automatically shut down. Alternatively, a developer can handle license initialization failure using the ILicenseControl *interface members to obtain information on the nature of the failure before the application is programmatically shut down.*

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 don't have them installed, rerun the Developer Kit Install wizard, click Custom or Modify, and click 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.

On Windows you can find this sample in:

```
<install_location>\DeveloperKit\samples\Developer_Guide_Scenarios\
Building_an_ArcGIS_Controls_Map_Viewer_ApplicationJava.zip
```

On Solaris and Linux you can find the sample in:

```
<install_location>/developerkit/samples/Developer_Guide_Scenarios/
Building_an_ArcGIS_Controls_Map_Viewer_ApplicationJava.zip
```

PROJECT DESCRIPTION

This scenario demonstrates the steps required to create a GIS application for viewing preauthored ESRI map documents, or MXDs. 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 ToolbarBean and TOCBean
- Adding toolbar items to the ToolbarBean
- Creating a popup menu using ToolbarMenu
- Controlling Label Editing in the TOCBean component
- Drawing an overview rectangle on the MapBean
- Creating a custom tool
- Customizing the ToolbarBean
- 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 JavaBeans-compatible IDEs. Each component has certain properties and methods and is capable of firing events. Internally, the Java components use JNI 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 MapBean, PageLayoutBean, TOCBean, and ToolbarBean components are placed inside a *javax.swing.JFrame* container and interact with each other and with other ArcGIS Engine objects to provide GIS viewing capability. The MapBean serves as an overview map in the corner of the application, while the PageLayoutBean is the visual centerpiece.

The scenario starts with building a GUI using the BorderLayout 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, the components will be programmatically placed; this gives a better understanding of the code. For information on using the components as drag-and-drop beans, 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):

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, click Custom or Modify, and click the Java feature under ArcGIS Engine. In addition, for access to the Javadoc and other Java-specific documentation, click the Java 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 Software Development Kit, preferably 1.4.2 or later. If you don't already have one available, download it from the Java Web site 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.

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, click Custom or Modify, and click the samples feature under Software Developer Kit.

- Access to the sample data and solution code that comes with this scenario.

  `<install_location>\DeveloperKit\samples\Developer_Guide_Scenarios\Building_an_ArcGIS_Controls_Map_Viewer_ApplicationJava.zip`

To build this application, the following visual components from the ArcGIS Engine Developer Kit will be used:

- map.MapBean
- pagelayout.PageLayoutBean
- TOC.TOCBean
- toolbar.ToolbarBean
- 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
- Control commands
- Display
- SystemUI

In the Java API, their package names are prefixed by 'com.esri.arcgis'.

To reference the above-mentioned packages, the following JAR files must be added to your class path:

- 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 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 doesn't 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, your development environment should be set up as described in the Java API section of Chapter 4, 'Developer environments'.

Use your favorite text editor or IDE to write your source code.

Building a GUI using the visual components

1. Create a new file called MapViewerFrame.java. The MapViewerFrame class will provide the GUI and functionality of the map viewer application. Implement this class as a subclass of *javax.swing.JFrame*.

```
// MapViewerFrame.java
import java.io.IOException;
import javax.swing.JFrame;

public class MapViewerFrame extends JFrame {
  // Constructor
  public MapViewerFrame() {
     setTitle("MapViewer");
  }
  public void buildAndShow() throws IOException {
     this.setVisible(true);
 }
} // end MapViewerFrame class
```

2. Create a new file called MapViewer.java. This will provide the *main* method, which will construct the MapViewerFrame, give it an initial size, and launch it:

```
// MapViewer.java
import java.awt.Dimension;
import java.awt.Toolkit;
```

```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.IOException;
import javax.swing.UIManager;

import com.esri.arcgis.system.AoInitialize;
import com.esri.arcgis.system.EngineInitializer;
import com.esri.arcgis.system.esriLicenseProductCode;

public class MapViewer {

  static {
    String laf = UIManager.getSystemLookAndFeelClassName();
    try {
       UIManager.setLookAndFeel(laf);
    } catch (Exception e) {
       // Ignore
    }
  }

   public static void main(String[] args) throws IOException {
    EngineInitializer.initializeVisualBeans();
    final AoInitialize aoInit = new AoInitialize();
    aoInit.initialize(
      esriLicenseProductCode.esriLicenseProductCodeEngine);

    MapViewerFrame mapViewerFrame = new MapViewerFrame();

    Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
    int x = d.width/6;
    int y = d.height/6;
    int width  = d.width*2/3;
    int height = d.height*2/3;
    mapViewerFrame.setBounds(x, y, width, height);
    mapViewerFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        try {
          aoInit.shutdown();
        } catch (IOException ex) {
          ex.printStackTrace();
        }
        System.exit(0);
      }
    });
    mapViewerFrame.buildAndShow();
  }
} // end MapViewer class
```

Any application that uses visual JavaBeans components of the ArcGIS Engine Developer Kit should include the following three JAR files in its class path:

…\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 class path 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 class path.

To compile using the command line, use the "cd" command to change to the directory containing the MapViewerFrame and MapViewer Java code and give a command similar to:

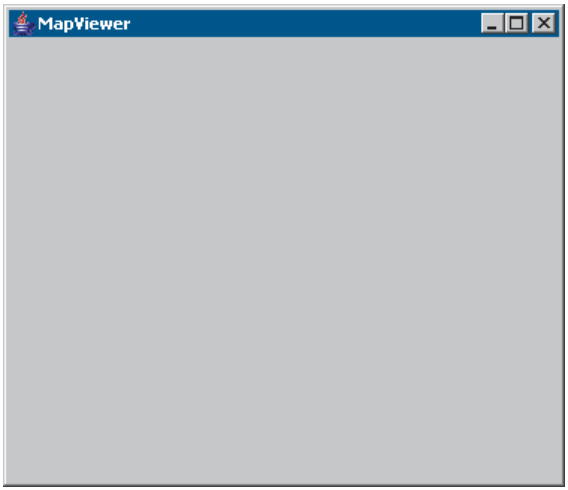
```
javac -classpath
  <ArcGIS_installation_directory>\java\opt\arcgis_visualbeans.jar;
  <ArcGIS_installation_directory>\java\opt\arcobjects.jar;
  <ArcGIS_installation_directory>\java\jintegra.jar;. *.java
```

This compiles the Java program and produces a MapViewerFrame.class and a MapViewer.class file. If you get NoClassDefFoundError, double-check your class path and make sure it includes all three required JAR files—arcobjects.jar, arcgis_visualbeans.jar, and jintegra.jar.

To run the Java program, click the Run button in your IDE or give the following command from the command line:

```
java -classpath
<ArcGIS_installation_directory>\java\
opt\arcgis_visualbeans.jar;
<ArcGIS_installation_directory>\java\opt\
arcobjects.jar;
<ArcGIS_installation_directory>\java\
jintegra.jar;. MapViewer
```

4. Launch the MapViewer class and make sure a blank Java frame comes up before proceeding to the next step.



5. Next, add member variables for the components to be added to the MapViewerFrame class and create these components in the constructor.

```
// MapViewerFrame.java
import java.io.IOException;

import javax.swing.JFrame;

// 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.TOCBean;
import com.esri.arcgis.beans.map.MapBean;
import com.esri.arcgis.beans.pagelayout.PageLayoutBean;
import com.esri.arcgis.beans.toolbar.ToolbarBean;

public class MapViewerFrame extends JFrame {
   PageLayoutBean pageLayout;
```

```
  MapBean map;
  TOCBean toc;
  ToolbarBean toolbar;
  JLabel statusLabel;

    // Constructor
 public MapViewerFrame() {
   setTitle("MapViewer");
   pageLayout = new PageLayoutBean();
   map = new MapBean();
   toc = new TOCBean();
   toolbar = new ToolbarBean();
   statusLabel = new JLabel(" ");
```

6. In the MapViewerFrame user interface, the toolbar component will occupy BorderLayout.NORTH, and the pageLayout component will occupy the BorderLayout.CENTER position.

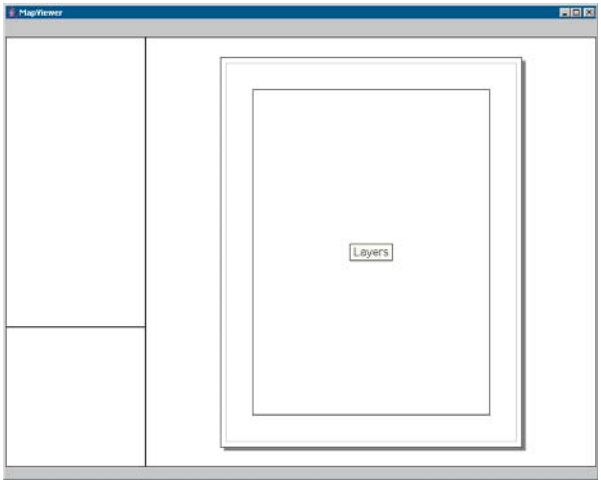
   The BorderLayout.WEST position will be occupied by both the TOC and map components. To achieve this, create a new method to build a JPanel containing the two controls with the desired layout:

```
// New method to build the left panel
private JPanel buildMapTOCPanel() {
  JPanel leftPanel = new JPanel();
  JPanel bottomPanel = new JPanel();
  leftPanel.setLayout(new BorderLayout());
  bottomPanel.setLayout(new BorderLayout());
  bottomPanel.setPreferredSize(new Dimension(200,200));
  bottomPanel.add(map, BorderLayout.CENTER);
  leftPanel.add(toc, BorderLayout.CENTER);
  leftPanel.add(bottomPanel, BorderLayout.SOUTH);
  return leftPanel;
}
```

7. 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.setVisible();
}
```

8. Compile and run the application to confirm that the GUI has been laid out as shown:



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 window and see that the components get updated 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 imports to MapViewerFrame.java:

```
import com.esri.arcgis.beans.pagelayout.IPageLayoutControlEventsAdapter;
import
com.esri.arcgis.beans.pagelayout.IPageLayoutControlEventsOnPageLayoutReplacedEvent;
```

2. Add the following code to the end of the buildAndShow method, just before the call to this.setVisible.

For cross-platform compatibility, the filenames and pathnames used to store data must be lowercased for Solaris and Linux.

```
public void buildAndShow() throws IOException {
  . . .
  . . .

addEventListeners();

// Load a preauthored map document into the PageLayout bean.
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);
this.setVisible(true);
```

2. When the document is loaded into the pageLayout bean, an onPageLayoutReplaced event is 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());
    }
  });
}
```

Buddying up the ToolbarBean and TOCBean

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 the control with which 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 preauthored 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 Beans
  // with the PageLayout Bean.
  toc.setBuddyControl(pageLayout);
  toolbar.setBuddyControl(pageLayout);
} this.setVisible(true);
```

Adding commands to the toolbar

The toolbar control has been added to the user interface. 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 MapBean, the PageLayoutBean, and the ToolbarBean. 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.

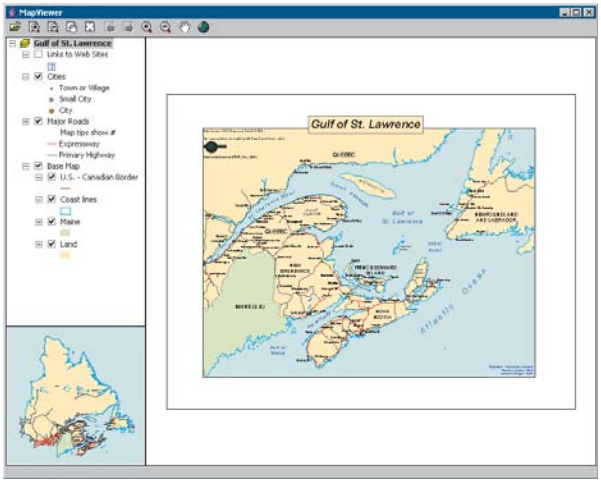
1. To add the prebuilt toolbar commands, add the following imports to MapViewerFrame.java:

```
import com.esri.arcgis.controlcommands.ControlsMapFullExtentCommand;
import com.esri.arcgis.controlcommands.ControlsMapPanTool;
import com.esri.arcgis.controlcommands.ControlsMapZoomInTool;
import com.esri.arcgis.controlcommands.ControlsMapZoomOutTool;
import com.esri.arcgis.controlcommands.ControlsOpenDocCommand;
import com.esri.arcgis.controlcommands.ControlsPagePanTool;
import com.esri.arcgis.controlcommands.ControlsPageZoomInTool;
import com.esri.arcgis.controlcommands.ControlsPageZoomOutTool;
import
com.esri.arcgis.controlcommands.ControlsPageZoomPageToLastExtentBackCommand;
import
com.esri.arcgis.controlcommands.ControlsPageZoomPageToLastExtentForwardCommand;
import com.esri.arcgis.controlcommands.ControlsPageZoomWholePageCommand;
import com.esri.arcgis.systemUI.esriCommandStyles;
```

2. In the buildAndShow() method, add the prebuilt commands to the toolbar before the addEventListeners() call:

```
// Add generic commands to the toolbar
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 preauthored 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 MapBean and PageLayoutBean. 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.controlcommands 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.IPageLayoutControlEventsOnMouseDownEvent;
import com.esri.arcgis.controlcommands.ControlsPageZoomInFixedCommand;
import com.esri.arcgis.controlcommands.ControlsPageZoomOutFixedCommand;
```

2. Add a member variable for the ToolbarMenu as shown below:

```
public class MapViewerFrame extends JFrame{

  PageLayoutBean pageLayout;
  MapBean map;
  TOCBean toc;
  ToolbarBean toolbar;
  ToolbarMenu popupMenu;
  JLabel statusLabel;
```

3. Construct the popupMenu object in a try/catch block:

```
// Constructor
public MapViewerFrame(){
      setTitle( "MapViewer" );
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      pageLayout = new PageLayoutBean();
      map = new MapBean();
      toc = new TOCBean();
      toolbar = new ToolbarBean();
      try {
         popupMenu = new ToolbarMenu();
      } catch (IOException e) {
         e.printStackTrace();
      }
       statusLabel = new JLabel( " " );
}
```

4. Add prebuilt commands from the com.esri.arcgis.controlcommands package to the popupMenu component, before adding generic commands in the buildAndShow() method:

```
// 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.
  . . .
```

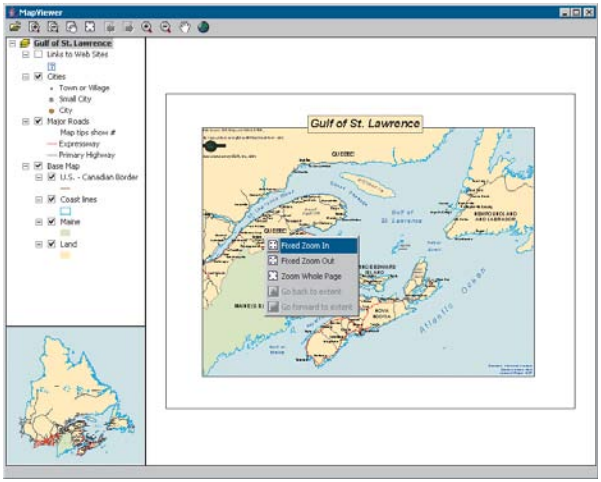
5. Associate the popupMenu with the pageLayout component, along with the code that sets up the buddy controls, in the buildAndShow() method:

```
// Set buddy controls to wire up the TOC and ToolbarBean
// with the pageLayout object.
toc.setBuddyControl(pageLayout);
toolbar.setBuddyControl(pageLayout);
popupMenu.setHook(pageLayout);


setVisible(true);
}
```

6. In the addEventListeners() method, add an event listener to the pageLayout component for the onMouseDown event, as an anonymous inner class, using IPageLayoutControlEventsAdapter:

```
pageLayout.addIPageLayoutControlEventsListener(
  new IPageLayoutControlEventsAdapter(){
  public void onPageLayoutReplaced(
     IPageLayoutControlEventsOnPageLayoutReplacedEvent evt)
     throws IOException{
      map.loadMxFile(pageLayout.getDocumentFilename(), null, null);
     map.setExtent(map.getFullExtent());
  }
  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 the pageLayout component to display the popup menu, and navigate around the page layout.

Controlling label editing in the TOCBean

By default, the TOCBean 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;
```

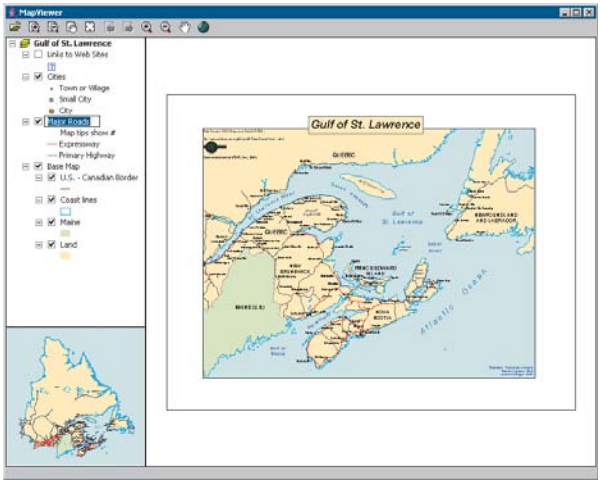
2. In the buildAndShow() method, set the labelEdit to manual:

```
. . .

// Set label editing to manual.
toc.setLabelEdit(esriTOCControlEdit.esriTOCControlManual);

// Add popup menu items.
. . .
```

3. In the addEventListeners() method, add an 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 it once, then click 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.



Drawing an overview rectangle on the MapBean

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 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.beans.pagelayout.
                    IPageLayoutControlEventsOnPageLayoutReplacedEvent;

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 MapBean
   Map focusMap;                // The PageLayoutBean's focus map

   PageLayoutBean 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 Bean. Add the following method:

```
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 MapBean.
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 below, 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 PageLayoutBean are the IPageLayoutControlEvents. These events do not tell you when the extent of the map within the data frame changes. To do this you will trap the ITransformEvents of the PageLayoutBean 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 *IEnvelope* to the new visible bounds of the map. By refreshing the MapBean you force it to redraw the shape on its display. Modify the listener you edited in the last step, as shown below:

```
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();
```

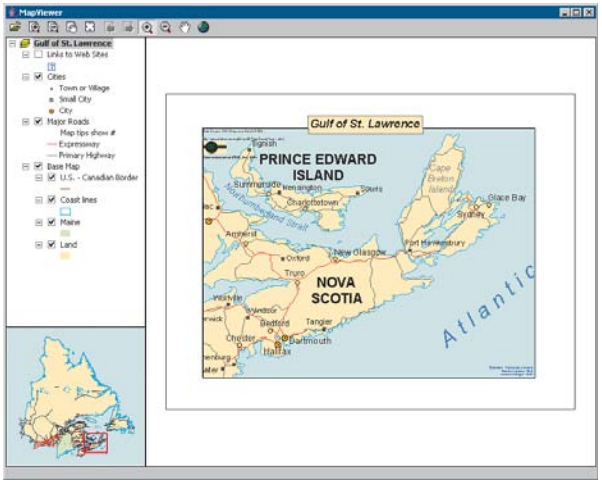
```
        DisplayTransformation dt = new
            DisplayTransformation(focusMap.getScreenDisplay().getDisplayTransformation());
        dt.addITransformEventsListener(new ITransformEventsAdapter(){

          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 MapBean.
        map.drawShape(currentExtent, fillSymbol);
      } catch (Exception e) {
        System.err.println("Error in drawing shape on
        MapBean");
        // 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.

Navigating around the focus map using the map navigation tools will change the extent of the focus map in the PageLayoutBean *and cause the* MapBean *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* PageLayoutBean*), so the* MapBean *will not update.*



Creating a custom tool

You will create a custom tool that can be added to the ToolbarBean. This tool will add today's date as a text element to the PageLayoutBean, at the location of a mouse click.

The tool will be built as a generic tool so it can work with MapBean and ToolbarBean as well as the PageLayoutBean.

Custom tools can be built as Java classes that implement the following two interfaces:

com.esri.arcgis.systemUI.ICommand;

com.esri.arcgis.systemUI.ITool;

1. Create a new file called AddDateCommand.java:

```
// AddDateCommand.java
import com.esri.arcgis.systemUI.ICommand;
import com.esri.arcgis.systemUI.ITool;

public class AddDateCommand implements ICommand, ITool {

} // end AddDateCommand class
```

Since 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.linar.jintegra.AutomationException;
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 {
    } // Ignore

} // end AddDateCommand class
```

3. The onCreate method is passed a reference to the control that the command works with. In this case, it can be a MapBean, PageLayoutBean, or ToolbarBean. 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 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 MapBean, PageLayoutBean, or ToolbarBean).

   Import the HookHelper class and add the *hookHelper* member variable to the *AddDateCommand* class:

```
import com.esri.arcgis.systemUI.ITool;
import com.esri.arcgis.controlcommands.HookHelper;

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
{
  // To be added later. Delete this line.
  hookHelper = new HookHelper();
  hookHelper.setHookByRef( obj );
}
```

5. Add implementation for all methods defined in the *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 onMouseDown 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");
```

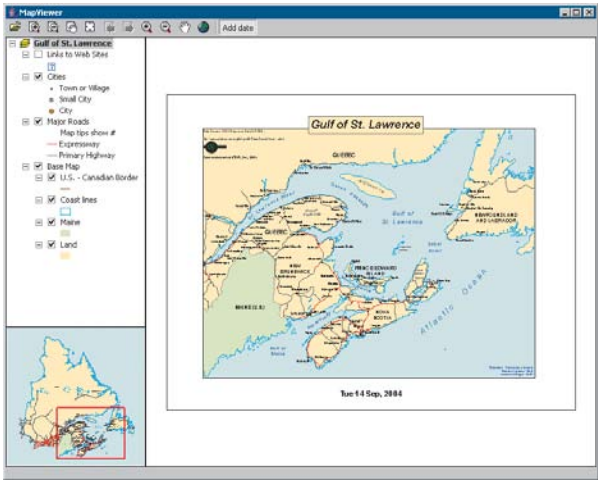
```
    String dateString = formatter.format(today);

    // 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 {
    // ignore
  }

  /*
   * 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 {
    // ignore
  }

  /*
   * see com.esri.arcgis.systemUI.ITool#onDblClick()
   */
  public void onDblClick() throws IOException, AutomationException {
    // ignore
  }
```

```
/*
 * see com.esri.arcgis.systemUI.ITool#onKeyDown(int, int)
 */
public void onKeyDown(int arg0, int arg1)
  throws IOException, AutomationException {
  // ignore
}

/*
 * see com.esri.arcgis.systemUI.ITool#onKeyUp(int, int)
 */
public void onKeyUp(int arg0, int arg1)
  throws IOException, AutomationException {
  // ignore
}

/*
 * 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 {
  // ignore
}

/*
 * 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 ControlsMapFullExtentCommand(), 0, -1, false, 0,
  esriCommandStyles.esriCommandStyleIconOnly);
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 the page layout to add today's date at that point.



Customizing the ToolbarBean

In addition to adding ArcGIS Engine commands and tools to the ToolbarBean using addItem() as shown above, you can also add them by customizing the ToolbarBean 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 to MapViewerFrame.Java:

```
import javax.swing.JCheckBox;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
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;
```

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.

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 ToolbarBean constructor.
```

3. Create a new method “createCustomizeDialog()” to instantiate the Customize dialog box. Add the following code:

```
private void createCustomizeDialog() throws IOException {
  customizeDialog = new CustomizeDialog();
  customizeDialog.setDialogTitle("Customize Toolbar Items");
  customizeDialog.setShowAddFromFile(true);
  // Set the toolbar 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:

```
public void buildAndShow() throws IOException {
  JPanel mapTOCPanel = buildMapTOCPanel();

  createCustomizeDialog();
  customizeCB = new JCheckBox("Customize");
```

5. In the addEventListeners method, add a listener to the customizeCB JCheckBox to start and close the Customize dialog box:

```
public void addEventListeners() throws IOException {
...
  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();
      }
    }
  });
...
```

6. To add the Customize check box to the GUI, modify the buildAndShow method 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.

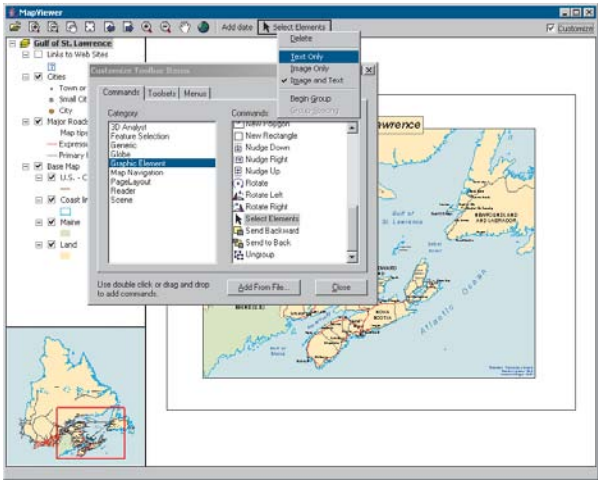
```
    JPanel topPanel = new JPanel();
    topPanel.setLayout(new BorderLayout());
    topPanel.add(toolbar, BorderLayout.CENTER);
    topPanel.add(customizeCB, BorderLayout.EAST);

    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);
```

7. In the *addEventListeners* method, add an event listener to the Customize dialog box to put the toolbar in the customize state when the dialog box is started and in the normal state when it is closed. Since these events are not generated by Java's event dispatching thread, modifications to the customizeCB check box should be made via the *SwingUtilities* class:

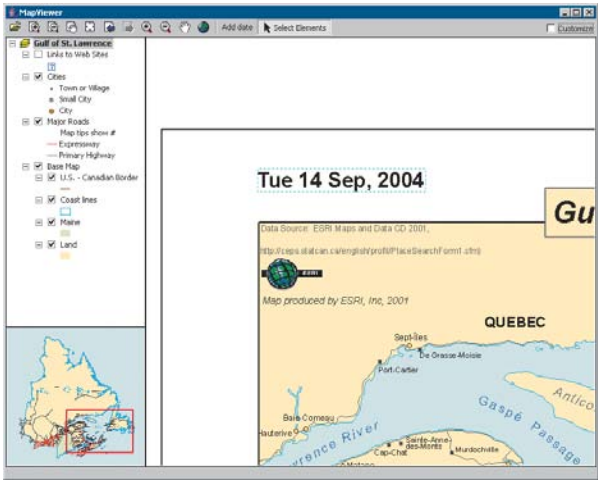
```
  private void addEventListeners() throws IOException {
    ...
    });
    customizeDialog.addICustomizeDialogEventsListener(new
      ICustomizeDialogEvents(){
```

```
        public void onStartDialog(ICustomizeDialogEventsOnStartDialogEvent
            arg0) throws IOException {
            toolbar.setCustomize(true);
        }
        public void onCloseDialog(ICustomizeDialogEventsOnCloseDialogEvent
            arg0) throws IOException {
            toolbar.setCustomize(false);
            java.awt.EventQueue.invokeLater (
            new Runnable () {
                public void run () {
                  customizeCB.setSelected(false);
                }
              }
            );
        }
});
```

8. Run the application and check the Customize check box to put the toolbar into customize mode and open the Customize dialog box.
9. 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 an item on the toolbar, you can adjust the appearance in terms of style and grouping.



10. Close the Customize Toolbar Item dialog box to 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: MapViewer
   ```
3. Make sure to press Enter at the end of the first line.
4. Save the file and open a command window. Use the 'cd' command to change 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 (*http://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, click Custom or Modify, and click 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\
Building_an_ArcGIS_Controls_Map_Viewer_ApplicationCSharp.zip
```

PROJECT DESCRIPTION

The goal of the 'Building applications with Windows Controls' scenario is to demonstrate and familiarize you 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, ToolbarControl, and LicenseControl 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 preauthored ESRI map documents, or MXDs. The scenario covers the following techniques:

- Loading and embedding the ArcGIS controls in Microsoft Visual Studio .NET.
- Loading preauthored map documents into 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.
- License configuration using the LicenseControl.
- 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 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, and the LicenseControl is used to configure the application with an appropriate license. 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 to highlight how the ArcGIS controls interact with each other and to expose a part of each ArcGIS control's 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, are advantageous.
- Access to the sample data and code that comes with this scenario. This is located at:

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, click Custom or Modify, and click the samples feature under Software Developer Kit.

<install_location>\DeveloperKit\samples\Developer_Guide_Scenarios\Building_an_ArcGIS_Controls_Map_Viewer_ApplicationCSharp.zip

The controls and libraries used in this scenario are as follows:

- ESRI.ArcGIS.AxMapControl
- ESRI.ArcGIS.MapControl
- ESRI.ArcGIS.AxPageLayoutControl
- ESRI.ArcGIS.PageLayoutControl
- ESRI.ArcGIS.AxTOCControl
- ESRI.ArcGIS.TOCControl
- ESRI.ArcGIS.AxToolbarControl
- ESRI.ArcGIS.ToolbarControl
- ESRI.ArcGIS.AxLicenseControl
- ESRI.ArcGIS.LicenseControl
- ESRI.ArcGIS.Carto
- ESRI.ArcGIS.System
- ESRI.ArcGIS.Display
- ESRI.ArcGIS.SystemUI
- ESRI.ArcGIS.Geometry
- ESRI.ArcGIS.Utility

The AxControlName .NET Framework components represent the control that is hosted within a .NET form, while the esriControlName *assemblies contain the object and interfaces from inside the control's type library.*

IMPLEMENTATION

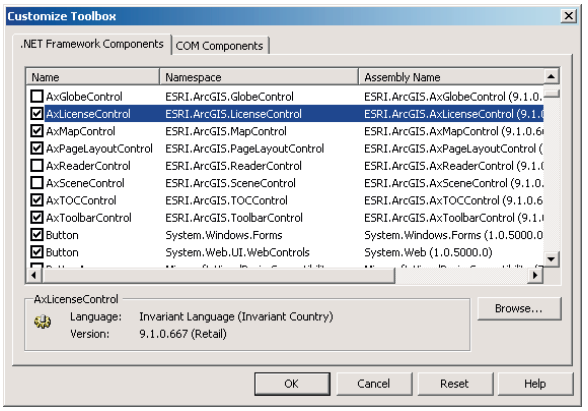
The implementation below provides you with all the code you will need to successfully complete the scenario. It does not provide step-by-step instructions to develop applications in Microsoft Visual Studio .NET, as it assumes that you have a working knowledge of the development environment already.

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.

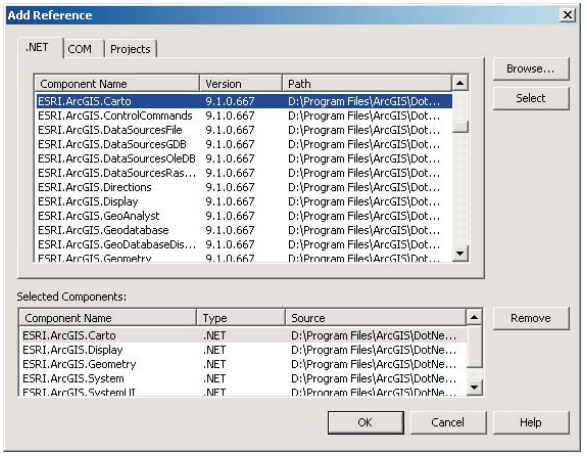
1. Start Visual Studio .NET and create a new Visual C# Windows Application project from the New project dialog box.
2. Name the project 'Controls', and browse to a location to save the project.

3. Right-click in the Windows Forms tab of the Toolbox and click Add/Remove Items from the context menu.
4. In the Customize Toolbox dialog box, click the .NET Framework Components tab and check AxMapControl, AxPageLayoutControl, AxTOCControl, AxToolbarControl, and AxLicenseControl. Click OK. The controls will now appear in the Windows Forms tab of the toolbox.



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.

5. Click on the Project menu and click 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, and ESRI.ArcGIS.Utility. Click OK.

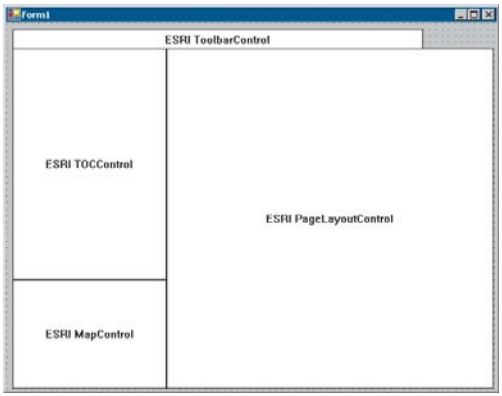


Embedding the ArcGIS controls in a container

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 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 esriToolbarControl;
using esriTOCControl;
```

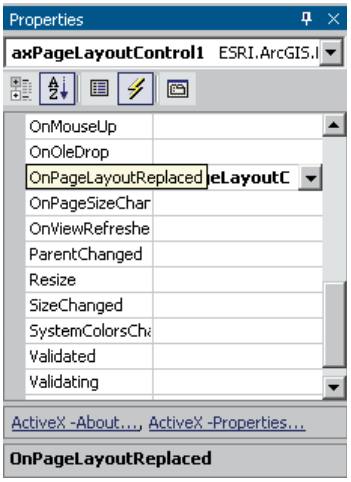
Loading map documents into the PageLayoutControl and MapControl

Individual data layers or preauthored map documents 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 preauthored 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 preauthored 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;
}
```

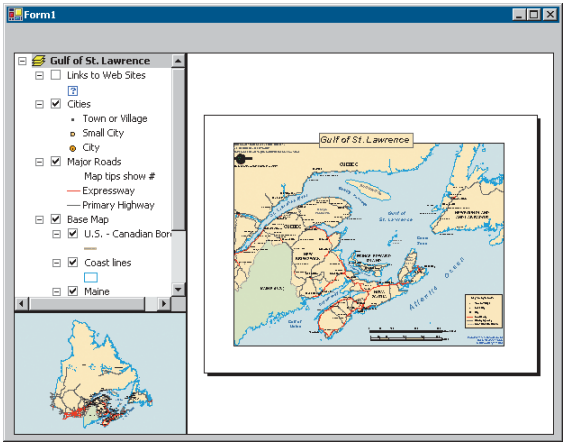
Setting the TOCControl and ToolbarControl buddy controls

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 preauthored 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);
}
```

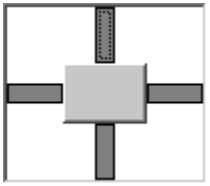
2. Build and run the application. The map document is loaded into the PageLayoutControl, and the TOCControl lists the data layers in the map document. Use the TOCControl to toggle layer visibility by checking and unchecking the boxes. By default, the focus map of the map document is loaded into the MapControl. 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.



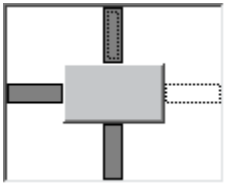
Handling form resize

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 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 you 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 you 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 run the application. Try resizing the form.

Adding commands to the ToolbarControl

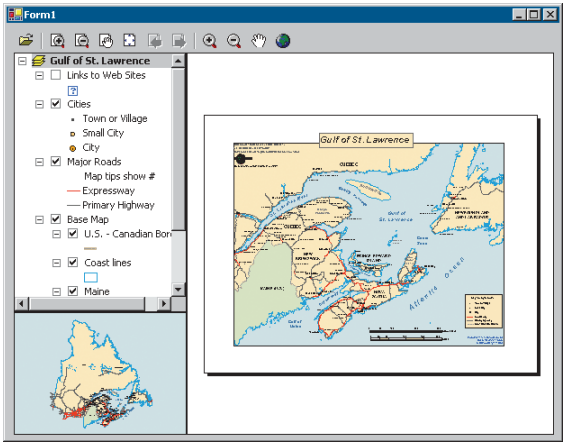
ArcGIS Engine comes with more than 120 commands and tools that work with the MapControl, the PageLayoutControl, and the ToolbarControl directly. These commands and tools provide you with a lot of frequently used GIS functionality for map navigation, graphics management, and feature selection. You will now add some of these commands and tools to your application.

1. 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 navigation 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 preauthored…
}
```

2. Build and 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.

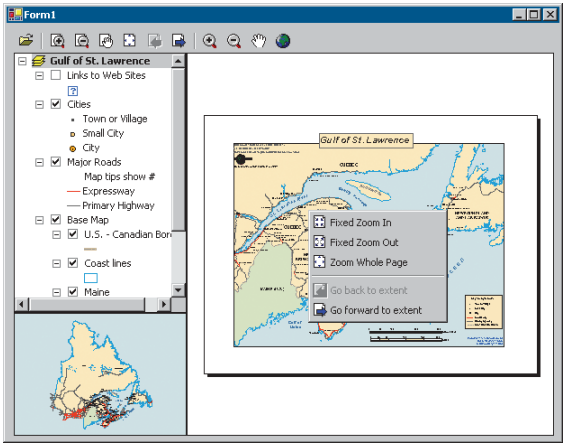
```
private void Form1_Load(object sender, System.EventArgs e)
{
  // Add Map navigation commands…

  // 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 preauthored…
}
```

3. Display the form in design mode, select axPageLayoutControl1 from the Properties window, and display the axPageLayoutControl events. Double-click 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 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.

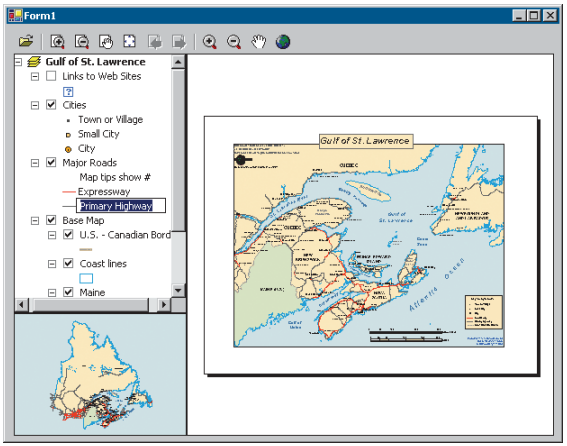
```
private void Form1_Load(object sender, System.EventArgs e)
{
  // Set label editing to manual.
  axTOCControl1.LabelEdit = esriTOCControlEdit.esriTOCControlManual;

  // Add generic commands.
}
```

2. Display the form in design mode, select AxTOCControl1 from the Properties window, and display the AxTOCControl events. Double-click OnEndLabelEdit to add an event handler to the code window.

3. Add the following code to the axTOCControl_OnEndLabelEdit 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 it once, then click 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

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.*

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.

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 you when the extent of the map within the data frame changes. To enable this functionality you will

use the *ITransformEvents* interface of the PageLayoutControl focus map. Add the following code to the PageLayoutControl_OnPageLayoutReplaced event handler directly above the load document code.

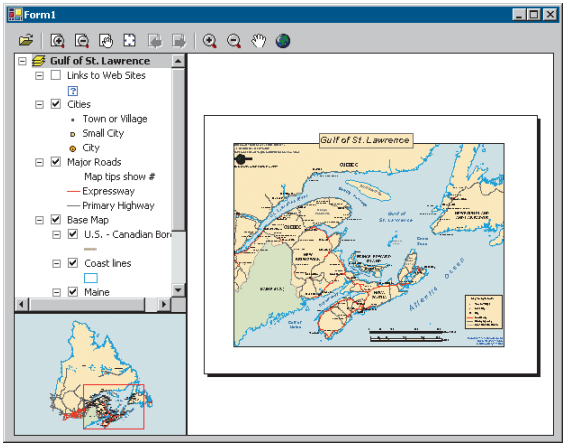
```
private void axPageLayoutControl1_OnPageLayoutReplaced(object sender,
ESRI.ArcGIS.PageLayoutControl.IPageLayoutControlEvents_OnPageLayoutReplacedEvent
e)
{
  // Get the IActiveView of the focus map in the PageLayoutControl.
  IActiveView activeView = (IActiveView)
axPageLayoutControl1.ActiveView.FocusMap;
  // Trap the ITransformEvents of the PageLayoutControl'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 preauthored 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 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\Building_an_ArcGIS_Controls_Map_Viewer_ApplicationCSharp.zip.

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 click 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 the Project menu and click Add Existing Item. Browse to the date.bmp file from its location in this sample's source code and add it into your project.
7. Click the date.bmp file 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 the project in the Solution Explorer and select Properties. In the project Property Pages, select General and change the Root Namespace. Click OK.

8. Change the namespace of 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 no 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.

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.

10. Specify that the AddDateTool class inherits from the ESRI BaseTool abstract class. Also add the *sealed* class modifier.

```
public sealed class AddDateTool : BaseTool
```

11. Add the following code to AddDateTool class constructor:

```
public AddDateTool()
{
  // Get an array resources in the assembly.
  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";
}
```

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.

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. Click Add, then Override to add the property to the code window.

14. Add the following code to override the OnCreate method.

```
public override void OnCreate(object hook)
{
  m_HookHelper.Hook = hook;
}
```

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.

15. In the Class View window, navigate to the BaseCommand Enabled property and right-click to display the context menu. Click Add, 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.

```
public override bool Enabled
{
  get
  {
    // Set the enabled property.
    if (m_HookHelper.ActiveView != null)
    {
      return true;
    }
    else
    {
      return false;
    }
  }
}
```

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 a MapControl, PageLayoutControl, or ToolbarControl).

17. In the Class View window, navigate to the BaseTool OnMouseDown method and right-click to display the context menu. Click Add, 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 element's 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 the Commands project and select Properties from the context menu.

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.

20. In the project Property Pages dialog box, select Configuration Properties, then click Build. In the right pane, change the Register for COM Interop property to True. Click OK.

If the Register for COM Interop property is disabled, check that the project is a C# class library type.

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")]
```

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 brackets.

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)
```

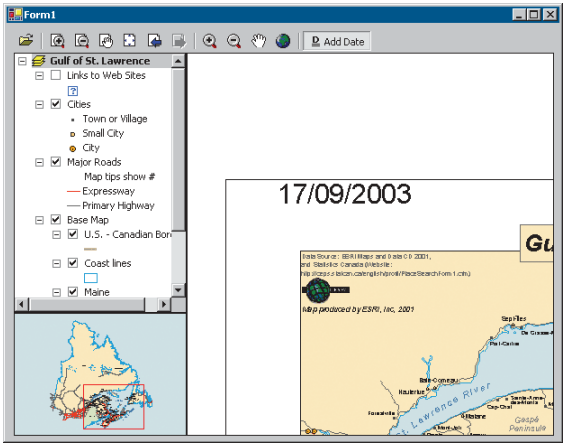
```
  {
      regKey.DeleteSubKey("{B284D891-22EE-4F12-A0A9-B1DDED9197F4}");
  }
}
#endregion
```

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 navigation 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

In addition to 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:

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.

```
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
```

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:

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.

```
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);
}
```

3. Call the CreateCustomizeDialog function from the Form_Load event before the call to the CreateOverviewSymbol subroutine.

```
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 CheckedChanged to add an event handler to the code window.
6. Add the following code to the chkCustomize_ CheckedChanged event.

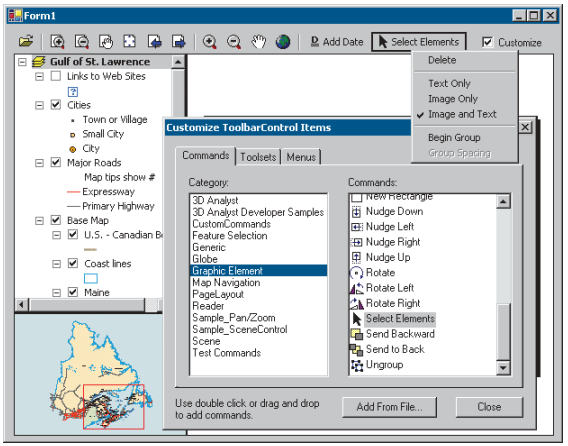
```
private void chkCustomize_CheckedChanged(object sender, System.EventArgs
e)
{
  // Show or hide the Customize dialog box.
  if (chkCustomize.Checked == false)
  {
    m_CustomizeDialog.CloseDialog();
    axToolbarControl1.Customize = false;
  }
  else
  {
    m_CustomizeDialog.StartDialog(axToolbarControl1.hWnd);
    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.

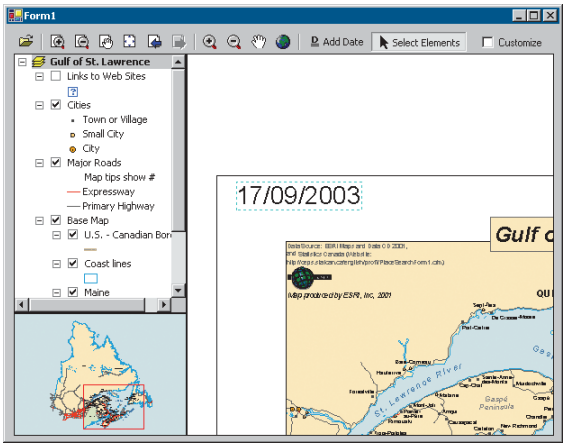
```
private void OnStartDialog()
{
  axToolbarControl1.Customize = true;
}

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 the Select Elements command to add it to the ToolbarControl. By right-clicking 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.



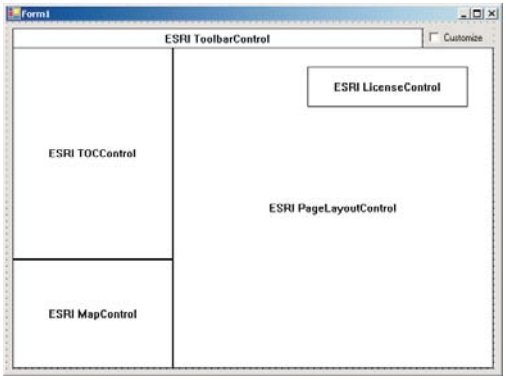
When developing a standalone executable using ArcObjects, it is the responsibility of the application to check and configure the licensing options. A license can be configured, using either the LicenseControl or the coclass AoInitialize *and the* IAoInitialize *interface it implements, that is designed to support license configuration. License initialization must be performed at application start time, before any ArcObjects functionality is accessed. Failure to do so will result in application errors. For more information about licensing see Chapter 5,'Licensing and deployment'.*

The LicenseControl will appear on a form at design time so that it can be selected and its property pages viewed. However, at runtime the LicenseControl is invisible so its position on the form is irrelevant.

LICENSE CONFIGURATION AND 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. License configuration can be performed either using the LicenseControl or programmatically using the *AoInitialize* object. For the purpose of this application, the LicenseControl will be used to handle license configuration.

1. Open the .Net form in design mode.
2. Double-click the AxLicenseControl in the Windows Forms tab of the toolbox to add a LicenseControl onto the form.

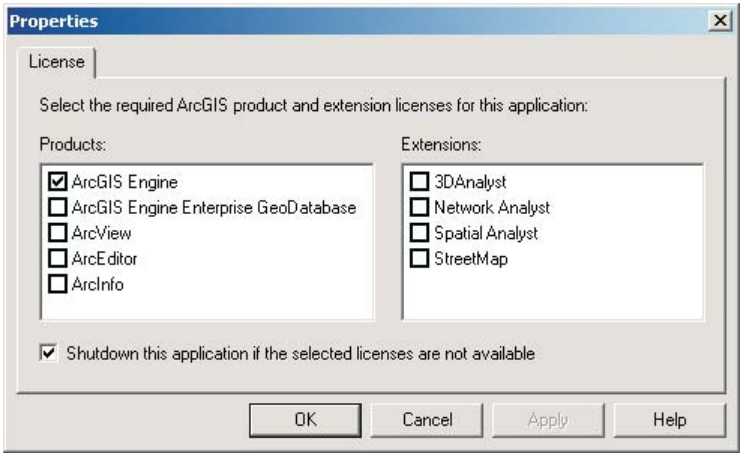


3. Right-click on LicenseControl and click Properties to open the LicenseControl property pages.

This application can be initialized with an ArcGIS Engine license, but you may optionally initialize the application with a higher product license. For example, if you check the 'ArcGIS Engine' license and the 'ArcView' license, the LicenseControl will initially try to initialize the application with an ArcGIS Engine license (the lower license). If that license is not available, the LicenseControl will try to initialize the application with an ArcView license (the next higher level license checked). If no product licenses are available, then the application will fail to initialize.

In this application the LicenseControl will handle license initialization failure. If the application cannot be initialized with an ArcGIS Engine product license, a License Failure dialog box will be displayed to the user before the application is automatically shut down. Alternatively, a developer can handle license initialization failure using the ILicenseControl *interface members to obtain information on the nature of the failure before the application is programmatically shut down.*

4. Check the ArcGIS Engine product license and check 'Shutdown this application if the selected licenses are not available'. Click OK.



5. Select Form1 from the Properties window and display the Form events. Double-click the Closing event to add an event handler to the code window.

6. In the Form_Closing event add the following code:

```
private void Form1_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
{
  // Release COM objects.
  ESRI.ArcGIS.Utility.COMSupport.AOUninitialize.Shutdown();
}
```

7. Build the project and build the solution in release mode.

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 Assembly Registration tool (RegAsm.exe) must be used to add information about the custom class to the registry.
- The user's machine will need an installation of the ArcGIS Engine Runtime and a standard ArcGIS Engine license.
- The user's 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.

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, click Custom or Modify, and click the samples feature under Software Developer Kit.

Motif and GTK ArcGIS C++ control widgets are only available on Solaris and Linux. However, GUI applications can be built on Windows with the COM API, including Visual C++, and the ActiveX ArcGIS controls.

The scenario has also been implemented with the GTK widget ArcGIS controls. That code can be found in the ArcGIS Engine Developer Kit.

This scenario is designed to introduce the ArcGIS Engine C++ API Control widgets. Its purpose is not to teach how to set up a C++ environment, how to compile using the make utility, or how to program with the Motif toolkit. Throughout this scenario it is assumed that the developer has a functional C++ environment and knows how to compile a C++ and Motif program in that environment. What this scenario does provide are the steps to take and the code to write to create a standalone ArcGIS controls application for viewing map documents, complete with a popup menu and custom tool.

You can find this sample in <install_location>/developerkit/samples/Developer_Guide_Scenarios/Building_an_ArcGIS_Controls_Map_Viewer_ApplicationMotif_Cpp.zip.

Although only C++ and Motif code samples are shown here, GIMP Toolkit (GTK) C++ code has also been written and can be found at <install_location>/developerkit/samples/Developer_Guide_Scenarios/Building_an_ArcGIS_Controls_Map_Viewer_ApplicationGTK_Cpp.zip.

PROJECT DESCRIPTION

The 'Building applications with C++ and control widgets' scenario demonstrates the steps required to develop and deploy a GIS application using the standard ArcGIS controls within the ArcGIS Engine C++ API. This scenario uses the MapControl, PageLayoutControl, TOCControl, and ToolbarControl as Motif widgets in code written using a standard text editor and compiled using the make utility.

This scenario demonstrates the steps required to create a GIS application for viewing preauthored ESRI map documents, or MXDs. The scenario covers the following techniques:

- Programming with the ArcGIS Engine in a standard text editor
- Programmatically placing the ArcGIS Engine controls in Motif widget forms
- Loading preauthored map documents into the MapControl and PageLayoutControl
- Setting ToolbarControl and TOCControl buddy controls
- 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 on a Solaris or Linux platform

CONCEPTS

This scenario is implemented using a text editor, the make utility, and Motif widget ArcGIS controls.

Each Motif ArcGIS control widget has a corresponding GTK ArcGIS control widget.

The ArcGIS Engine C++ API provides reusable Motif widget components corresponding to each ArcGIS Engine control. This developer scenario will show how these components can be embedded in a Motif form to build a map viewer application.

Motif itself is a specification for how graphical user interfaces should look and feel. The Open Software Foundation (OSF) Motif Toolkit, used with the ArcGIS Engine C++ API in this scenario, uses X as the window system and X Toolkit Intrinsics as the API platform. The Motif widgets discussed here are one part of the Motif toolkit. Each widget is a reusable and configurable component of the user interface. The Motif toolkit provides widgets for common tasks, such as the toggle button used in this scenario. Each built-in Motif widget knows how to draw itself as well as some generic behavior; for example, the toggle button knows to call a function when it is clicked. However, it is up to the programmer to implement the callback function, giving the widget its specific behavior. In addition to providing widgets the Motif toolkit does other work for the developer, including handling many user interactions, managing window layout, and redrawing.

The Motif widget ArcGIS controls are custom widgets made available through the C++ API of the ArcGIS Engine Developer Kit. Placed in a top-level application widget, the controls have events, properties, and methods that the developer can access, and like the built-in Motif toolkit widgets, each ESRI control widget knows how to draw itself. The objects and functionality within each control can be combined with other ESRI ArcObjects as well as with custom controls to easily create customized end user applications.

This scenario was written in C++ using the Motif toolkit from the Open Software Foundation. It was chosen to create an application that would run on a Solaris or Linux platform. The same application could also be written in Java. Whichever API you use, your future success with the ArcGIS controls depends on your skill in both the language and ArcObjects.

DESIGN

This scenario has been designed to highlight how the ArcGIS controls interact with each other and to expose a part of each ArcGIS control's object model to the developer.

A buddy control is a control that is designed to work in conjunction with another control. For example, the Table of Contents, or TOC, Control is a buddy of the MapControl.

Although the GUI design will be different for GTK developers, the majority of the rest of the application code will be the same. Throughout the scenario, sidebars will highlight areas where GTK programmers will take different steps.

The scenario starts with building a GUI using Motif to place and manage the controls. The controls are then connected to each other through the *SetBuddy* methods. At this stage, the application is ready to function as a simple map viewer. The GUI functionality is then extended through a custom tool and event handling. To achieve this, the scenario further explores the API of the visual Motif widgets, as well as the other nonvisual ArcGIS Engine components.

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 ESRI ArcGIS Engine Developer Kit with an authorization file enabling it for development use.

- Your favorite text editor.
- A supported C++ compiler.
  - o Solaris: Sun WorkShop (Forte) 6 update 2
  - o Linux: GCC version 3.2
- A configured C++ environment (for example, your compiler configured and your path set). To test this, see 'Pre-ArcObjects C++ configuration steps' in Chapter 4, 'Development environments'.
- A configured ArcObjects environment. To get your machine ready for ArcObjects development, you must source the file <install_location>/init_engine.sh (or .csh, depending on your shell of choice). If you prefer, that can be done in your shell's RC file. Otherwise, you must source that file for each shell.
- Familiarity with your operating system and a basic foundation in both C++ and Motif programming. Although this scenario uses the Motif toolkit, it is not intended to teach the basics of Motif programming. For Motif-specific details, see the *Motif Programming Manual* resource listed at the end of this example.
- 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, are 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, click Custom or Modify, and click the samples feature under Software Developer Kit.

- Access to the sample data, code, and makefiles that come with this scenario. Code and makefiles are located at <install_location>/developerkit/Building_an_ArcGIS_Controls_Map_Viewer_ApplicationMotif_Cpp.zip. The sample data is located at <install_location>/developerkit/samples/data/arcgis_engine_developer_guide.

The GTK implementation can be found in the same directory in Building_an_ArcGIS_controls_Map_Viewer_ApplicationGTK_cpp.zip.

The controls and ArcGIS libraries used in this scenario are as follows:

- MapControl
- PageLayoutControl
- TOCControl
- ToolbarControl
- Carto Object Library
- System Object Library
- Display Object Library
- SystemUI Object Library
- Geometry Object Library

In GTK C++ you would include ArcSDK.h and Ao/AoGTKControls.h.You would use libarcsdk.so, libgtkctl.so, and libaoctl.so. GTK developers will also need to link against some GTK libraries and include some GTK-specific directories. It is recommended that pkg-config be used for this. These steps are shown in the provided Makefile.SolarisGTK.Template and Makefile.LinuxGTK.Template.

In Motif C++, you must include the following files to use those controls and object libraries:

- ArcSDK.h
- Ao/AoMotifControls.h

They also use the Engine Developer Kit libraries:

- libarcsdk.so
- libmotifctl.so
- libaoctl.so

In addition, for Motif you must link against:

- libpthread.so (Linux only)
- libXm.so
- libX11.so

To implement a custom tool, you must include the following file:

- Ao/AoToolBase.h

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 for developing C++ applications with Motif, as it assumes that you have a working knowledge of the development environment already.

During this scenario, the steps assume that you are programming on Solaris. However, to follow this same scenario on Linux all you will need to do is to use the Linux makefile, Makefile.LinuxMotif, instead of the Solaris makefile, Makefile.SolarisMotif.

Creating the makefile

To easily compile the application you make use of a makefile. This scenario is not designed to teach you the basics of project management with the make utility, so if you are unfamiliar with them, please see the Oram and Talbott reference at the end of this scenario.

Makefiles greatly simplify the build process for large applications. ESRI's Makefile samples don't always use all of the functionality available in the make utility (for example, SUFFIXES and pattern rules) because they are designed to work across many different versions of make. However, the makefiles do use various predefined variables that are available for most versions of make.These predefined variables include both commands (for example, CXX and RM) and command arguments (for example, CXXFLAGS and LDFLAGS). Consult the documentation for the make utility for more information about its advanced features.

1. To create the makefile for this scenario, copy either the Makefile.SolarisMotif.Template (for Solaris programming) or Makefile.LinuxMotif.Template (for Linux programming). Those files are located with the code in the Motif_Cpp folder. Remove the ".Template" from the end of the filename, and replace all instances of "motif_sample" with "MapViewer." The Makefile.SolarisMotif.Final and Makefile.LinuxMotif.Final, in the same directory, show what your makefile should look like at the end of this scenario.

Creating the Motif Application form by placing the ArcGIS Engine controls on a Motif Application form

To use the controls, you must create them as Motif widgets and place them into a Motif application form.

1. Open MapViewer.h, a new file, in your text editor. Place the following lines to ensure that the class is only declared once. The remainder of code for this file will fall between #define and #endif.

```
#ifndef __ENGINE_CONTROL_MOTIF_EXAMPLE__
#define __ENGINE_CONTROL_MOTIF_EXAMPLE__
```

```
#endif  // __ENGINE_CONTROL_MOTIF_EXAMPLE__
```

2. Include the necessary files for the Motif toolkit. Make sure that String, Cursor, Object, and ObjectClass have all been defined as ESRI types, as shown below:

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.

For GTK you will need to replace the Motif headers with a single include of gtk/gtk.h.

```
#define __ENGINE_CONTROL_MOTIF_EXAMPLE__

// Motif Headers
#define String       esriXString
#define Cursor       esriXCursor
#define Object       esriXObject
#define ObjectClass   esriXObjectClass
#include <Xm/Xm.h>
#include <Xm/MainW.h>
#include <Xm/Form.h>
#include <Xm/Protocols.h>
#undef String
#undef Cursor
#undef Object
#undef ObjectClass

#endif  // __ENGINE_CONTROL_MOTIF_EXAMPLE__
```

3. Include the Engine C++ API header file:

```
#undef ObjectClass

// ArcObjects Headers

// Engine
#include <ArcSDK.h>
```

4. Open MapViewer.cpp, a new file, in your text editor to implement your application. Include the MapViewer.h file just created.

```
#include "MapViewer.h"
```

5. Begin writing the *main* function. First initialize the ArcGIS Engine with *AoInitialize*, then set up the licensing for the product using IAoInitialize > Initialize(). If a user attempts to run the application without an appropriate ArcGIS Engine runtime or license, the application will exit. Notice that the scope is set to prevent ipInit from existing when ::AoUninitialize() is called later.

Although the names are similar, AoInitialize() and the IAoInitialize *interface have different purposes. While AoInitialize() is a C++ API call that initializes libraries, the* IAoInitialize *interface is used to handle licensing for the application.*

```
int main (int argc, char* argv[])
{
  // Initialize the engine
  ::AoInitialize(NULL);
  {
    IAoInitializePtr ipInit(CLSID_AoInitialize);
    esriLicenseStatus status;
    ipInit->Initialize(esriLicenseProductCodeEngine, &status);
    if (status != esriLicenseCheckedOut)
      AoExit(0);
  }
}
```

6. To increase code readability, use a helper function *FormSetup*. You will need to pass in an XtAppContext, which will be set in that function, as well as the parameters passed into *main*.

   a. First, place a definition of this file in MapViewer.h. You will also need to inform the compiler where to find the Resize function, which you will be using in FormSetup().

```
#include <ArcSDK.h>

void FormSetup(int argc, char* argv[], XtAppContext* app_context);
extern "C" void XtResizeWidget(Widget, _XtDimension,
                               _XtDimension, _XtDimension);

#endif  // __ENGINE_CONTROL_MOTIF_EXAMPLE__
```

   For the equivalent GTK setup, see `form_setup` *in the provided MapViewer.cpp file in the GTK zip file. You will notice that the GTK function names differ slightly from the Motif ones. This is to maintain the programming styles associated with each.*

   b. After initialization in MapViewer.cpp, call *FormSetup*, which you will write in the following steps.

```
::AoInitialize(NULL);
{
  …
}

XtAppContext app_context;
FormSetup(argc, argv, &app_context);
```

7. In MapViewer.cpp, after the *main* function, begin writing *FormSetup* by setting the language procedure for your application.

```
void FormSetup(int argc, char* argv[], XtAppContext* app_context)
{
  XtSetLanguageProc(NULL,NULL,NULL);
}
```

8. Continuing in *FormSetup*, create the Motif form for the MapViewer.

   a. Initialize the Motif toolkit and create your top-level application Motif widget.

```
XtSetLanguageProc(NULL,NULL,NULL);

// Initialize the Motif toolkit and create the parent widget
Widget topLevel = XtVaAppInitialize(app_context, "XApplication", NULL,
                                    0, &argc, argv, NULL, NULL);
XtVaSetValues(topLevel, XmNtitle, "MapViewer", NULL);
```

   b. Set the application's initial size to 800 x 640, using the resize function that you placed in MapViewer.h earlier.

```
XtVaSetValues(topLevel, XmNtitle, "MapViewer", NULL);

// Set the application size by resizing the created widget
XtResizeWidget(topLevel, 800, 640, 1);
```

   c. Create the main application window after the resize. Create the *main* form and attach it to the *main* window so that it fills the window. The *main* form widget will be the parent widget for each of the ESRI Engine Control widgets.

```
XtResizeWidget(topLevel, 800, 640, 1);

Widget mainWindow = XtVaCreateWidget("mainwindow",
                                     xmMainWindowWidgetClass, topLevel,
                                      NULL);

Widget mainForm = XtVaCreateWidget("mainform",
                                     xmFormWidgetClass,   mainWindow,
                                     XmNtopAttachment,    XmATTACH_WIDGET,
                                     XmNtopWidget,        mainWindow,
                                     XmNbottomAttachment, XmATTACH_WIDGET,
                                     XmNbottomWidget,     mainWindow,
                                     XmNleftAttachment,   XmATTACH_WIDGET,
                                     XmNleftWidget,       mainWindow,
                                     XmNrightAttachment,  XmATTACH_WIDGET,
                                     XmNrightWidget,      mainWindow,
                                     XmNfractionBase,     100,
                                     NULL);
```

d. Manage the child widgets.

```
Widget mainForm = XtVaCreateWidget(…);

// Manage the non-parent widgets
XtManageChild(mainWindow);
XtManageChild(mainForm);
```

e. Listen for window manager events to exit the application on command. To set this up, you will use a callback on the entire application that will respond to window manager protocols. When the window manager's close message is received, the *CloseAppCallback* function is executed. Here just listen—you will implement the callback function in Step 9.

```
XtManageChild(mainForm);

// Handle the "close" window manager message
Atom wm_delete_window = XmInternAtom(XtDisplay(topLevel),
"WM_DELETE_WINDOW", FALSE);
XmAddWMProtocolCallback(topLevel,
                        wm_delete_window, CloseAppCallback,
                        NULL);
```

GTK deals with closing differently, as shown in `form_setup` *of GTK's MapViewer.cpp.*

f. With all the application widgets created and managed and all the callbacks registered, the application can start running. Realizing the top-level widget recursively creates the actual windows for all the application widgets. The call to the function is made at the end of your *FormSetup* function.

```
void FormSetup(…)
{
…

 XmAddWMProtocolCallback(topLevel,
                         wm_delete_window, CloseAppCallback,
                         NULL);
```

```
    //Start the application running
    XtRealizeWidget(topLevel);
  }
```

9. In the previous step you set up a callback to listen for window manager protocols to close on command. Now implement the callback for when the application is closed.

   a. First give a forward declaration of the function. This is placed right after the declaration for *XtResizeWidget* at the top of MapViewer.h.

```
extern "C" void XtResizeWidget(Widget, _XtDimension,
                               _XtDimension, _XtDimension);
void CloseAppCallback(Widget w, XtPointer client_data,
                      XtPointer call_data);
```

   b. Write the function in MapViewer.cpp after *FormSetup*. Shut down and uninitialize the ArcGIS Engine Developer Kit, then exit.

```
// Function called when WM_DELETE_WINDOW protocol is passed
void CloseAppCallback(Widget w, XtPointer client_data,
                      XtPointer call_data)
{
  // Uninitialize the engine
  {
   IAoInitializePtr ipInit(CLSID_AoInitialize);
   ipInit->Shutdown();
  }
 ::AoUninitialize();

 AoExit(0);
}
```

Although it might look like a new instance of AoInitialize *is created, it is a singleton object, so this returns a pointer to the same* AoInitialize *object created before.*

10. A final ArcGIS Engine C++ API call is required to turn the application over to the X Toolkit Intrinsics, which handles passing events to the widgets. After this call, the application code sits idle and waits for user-generated events. This will end your *main* function.

```
int main(int argc, char* argv[])
{
  …

  FormSetup(argc, arcv, &app_context);

  // Start the application running
  XtAppMainLoop(app_context);
}
```

The GTK equivalent is `gtk_main`.

11. Compile the application by typing "make –f Makefile.SolarisMotif" at the command line.

12. Run the application by typing either "make –f Makefile.SolarisMotif run" or "./MapViewer" at the command line. You will see an empty form titled MapViewer.

Placing the ArcGIS Engine Controls

Now that you have a Motif form ready, you can create the ArcGIS controls as Motif widgets and place them on the form.

1. Start by including the ArcGIS controls header file in MapViewer.h:

```
#include <ArcSDK.h>

// Controls
#include <Ao/AoMotifControls.h>
```

2. In MapViewer.cpp, continue implementing your application. Set up global variables for the control interfaces, placing them before *main*.

```
// Control Interfaces
IToolbarControlPtr g_ipToolbarControl;
IMapControl3Ptr g_ipMapControl;
ITOCControlPtr g_ipTOCControl;
IPageLayoutControlPtr g_ipPageLayoutControl;

int main(int argc, char* argv[])
```

3. You are now ready to create the ESRI control widgets for the PageLayoutControl, MapControl, TOCControl, and ToolbarControl. You will do this in the FormSetup function after the *mainForm* is created and before the widgets are managed. The widget class for all of the ESRI controls is

mwCnlWidgetClass, and each widget must be given the MwNProgID that corresponds to its control type. Place the MapControl and the TOCControl in their own frames. Set the widget attachments to position the controls into the application window so that the ToolbarControl is along the top of the application, the TOCControl is along the left with the MapControl below it, and the PageLayoutControl is to the right of the TOCControl. Also set the height of the toolbar, the width of the TOC, and the dimensions of the map, which you want constant even if the application is resized.

```
Widget mainForm = XtVaCreateWidget(…);
```

GTK ArcGIS control widgets are created with `gtk_axctl_new`.

```
// ToolbarControl setup
Widget toolbarWidget = XtVaCreateWidget("toolbarwidget",
                                        mwCtlWidgetClass, mainForm,
                                        XmNtopAttachment, XmATTACH_FORM,
                                        XmNleftAttachment, XmATTACH_FORM,
                                        XmNrightAttachment, XmATTACH_FORM,
                                        MwNprogID, AoPROGID_ToolbarControl,
                                        NULL);
XtVaSetValues(toolbarWidget, XmNheight, 25, NULL);

// Create a sub-form to place TOCControl and MapControl on
Widget leftFormPanel = XtVaCreateWidget("leftformpanel",
                                       xmFormWidgetClass, mainForm,
                                       XmNtopAttachment, XmATTACH_WIDGET,
                                       XmNtopWidget, toolbarWidget,
                                       XmNbottomAttachment, XmATTACH_FORM,
                                       XmNleftAttachment, XmATTACH_FORM,
                                       XmNwidth, 200,
                                       NULL);

// MapControl setup
Widget mapWidget = XtVaCreateWidget("mapwidget",
                                    mwCtlWidgetClass,     leftFormPanel,
                                    XmNbottomAttachment,  XmATTACH_FORM,
                                    XmNleftAttachment,    XmATTACH_FORM,
                                    XmNrightAttachment,   XmATTACH_FORM,
                                    MwNprogID,      AoPROGID_MapControl,
                                    NULL);
XtVaSetValues(mapWidget, XmNheight, 200, XmNwidth, 200, NULL);

// TOCControl setup
Widget tocWidget = XtVaCreateWidget("tocwidget",
                                    mwCtlWidgetClass,     leftFormPanel,
                                    XmNtopAttachment,     XmATTACH_FORM,
                                    XmNleftAttachment,    XmATTACH_FORM,
                                    XmNrightAttachment,   XmATTACH_FORM,
                                    XmNbottomAttachment, XmATTACH_WIDGET,
                                    XmNbottomWidget,      mapWidget,
                                    MwNprogID,    AoPROGID_TOCControl,
                                    NULL);
XtVaSetValues(tocWidget, XmNwidth, 200, NULL);
```

```
// PageLayoutControl setup
Widget pageWidget = XtVaCreateWidget("pagewidget",
                                     mwCtlWidgetClass,      mainForm,
                                     XmNtopAttachment,   XmATTACH_WIDGET,
                                     XmNtopWidget,         toolbarWidget,
                                     XmNleftAttachment,  XmATTACH_WIDGET,
                                     XmNleftWidget,        leftFormPanel,
                                     XmNbottomAttachment,  XmATTACH_FORM,
                                     XmNrightAttachment,   XmATTACH_FORM,
                                     MwNprogID, AoPROGID_PageLayoutControl,
                                     NULL);

// Manage the non-parent widgets
```

4. Once the widgets are created, you can get an interface pointer to each control through Ao<ControlName>ControlGetInterface. Throughout this scenario smart pointers are used.

The GTK controls get interface pointers with `gtk_axctl_get_interface`.

```
// ToolbarControl setup
Widget toolbarWidget = XtVaCreateWidget(…);
XtVaSetValues(toolbarWidget, XmNheight, 25, NULL);
MwCtlGetInterface(toolbarWidget, (IUnknown**)&g_ipToolbarControl);

…

// MapControl setup
g_mapWidget = XtVaCreateWidget(…);
XtVaSetValues(mapWidget, XmNheight, 200, XmNwidth, 200, NULL);
MwCtlGetInterface(mapWidget, (IUnknown**)&g_ipMapControl);

// TOCControl setup
g_tocWidget = XtVaCreateWidget(…);
XtVaSetValues(tocWidget, XmNwidth, 200, NULL);
MwCtlGetInterface(tocWidget, (IUnknown**)&g_ipTOCControl);

// PageLayoutControl setup
g_pageWidget = XtVaCreateWidget(…);
MwCtlGetInterface(pageWidget, (IUnknown**)&g_ipPageLayoutControl);
```

5. Call *XtManageChild* on the Control widgets. The parent widget will take care of the size and placement.

```
XtManageChild(mainForm);
XtManageChild(leftFormPanel);
XtManageChild(toolbarWidget);
XtManageChild(mapWidget);
XtManageChild(tocWidget);
XtManageChild(pageWidget);
```

6. Since you are using the ArcGIS controls, you must use a new method for the *main* application loop. In *main*, change *XtAppMainLoop* to *MwCtlAppMainLoop*:

In GTK there is no ArcGIS-specific method for the main application loop, and `gtk_main` *continues to be used. However,* `gtk_axctl_initialize_message_queue` *must be used before* `gtk_main` *to enable MainWin message delivery.*

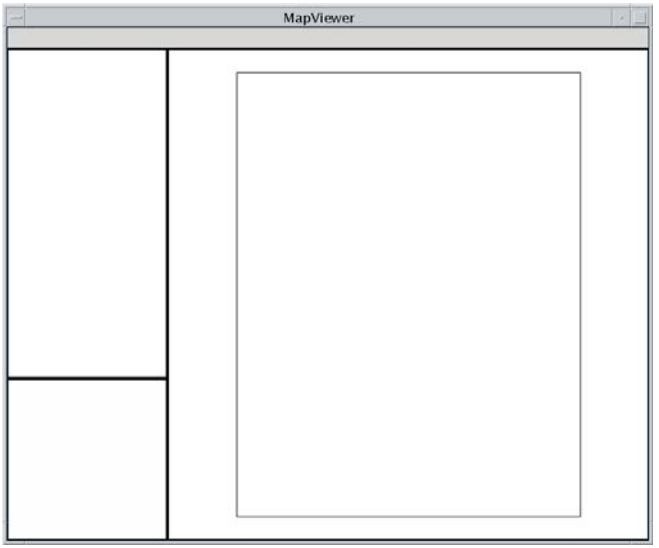
```
// Start the application running
XtAppMainLoop(app_context);
```

```
MwCtlAppMainLoop(app_context);
```

7. Before you shut down the application, you need to clean up the global ArcObjects. Do this by setting the pointer for each control to 0.

```
void CloseAppCallback(Widget w, XtPointer client_data,
                      XtPointer call_data)
{
  g_ipToolbarControl = 0;
  g_ipMapControl = 0;
  g_ipTOCControl = 0;
  g_ipPageLayoutControl = 0;

// Uninitialize the engine
```

8. Compile the application by typing "make –f Makefile.SolarisMotif" at the command line.
9. Run the application by typing either "make run –f Makefile.SolarisMotif" or "./MapViewer" at the command line. Notice how the controls have been placed in the application window. At this point the controls are all empty because no commands or data have been added. Try resizing the *main* form, and see that the TOCControl maintains its width, the ToolbarControl its height, and the MapControl its dimensions, but the other dimensions and controls resize themselves.



Loading map documents into the PageLayoutControl and MapControl

Individual data layers or preauthored ESRI map documents can be loaded into the MapControl and PageLayoutControl. You can either load the sample map document provided, or you can load your own map document. Later you will add an ArcGIS command to the ToolbarControl, which will allow you to browse to a map document.

1. Programmatically add data to the PageLayoutControl. To do so you will write a new function, *LoadData*.

   a. Place a forward declaration after the forward declaration for the *CloseAppCallback* function at the top of MapViewer.h.

   ```
   void CloseAppCallback(Widget w, XtPointer client_data,
                         XtPointer call_data);
   void LoadData();
   ```

   b. Now define *LoadData* in MapViewer.cpp after the definition for the *CloseAppCallback* function.

   ```
   void CloseAppCallback(Widget w, XtPointer client_data,
                         XtPointer call_data)
   {
     …
   }

   void LoadData()
   {
    CComBSTR MX_DATAFILE;
    MX_DATAFILE =
    L"../data/arcgis_engine_developer_guide/gulf of st. lawrence.mxd";
    VARIANT_BOOL bValidDoc;
    g_ipPageLayoutControl->CheckMxFile(MX_DATAFILE, &bValidDoc);
    if (bValidDoc)
      g_ipPageLayoutControl->LoadMxFile(MX_DATAFILE);
   }
   ```

   Data can also be loaded to the toolbar's buddy at runtime by using esriControlCommands.ControlsOpenDocCommand, a command that will be placed on the toolbar later in this scenario.

   c. Call *LoadData* from the *main* function after the widgets are placed in *FormSetup*.

   ```
   FormSetup(argc, argv, &app_context);

   LoadData();
   ```

2. You want the same map to appear in the MapControl. When the document in the PageLayoutControl changes, the contents of the MapControl must be updated. To do that, you must listen for events in the PageLayoutControl by writing a class that inherits from *IPageLayoutControlEventsHelper*. Open a new text file called PageLayoutControlEvents.h. Place the following code into that file, making sure to include the macro IUNKNOWN_METHOD_DEFS to implement *IUnknown*:

   ```
   #ifndef __PAGELAYOUTCONTROLEVENTS_H_
   #define __PAGELAYOUTCONTROLEVENTS_H_

   // ArcObjects Headers
   ```

By using Ao/AoControls.h, this class works with both GTK and Motif applications.

```
// Engine
#include <ArcSDK.h>
#include <Ao/AoControls.h>

class PageLayoutControlEvents : public IPageLayoutControlEventsHelper
{
public:

// IUnknown
IUNKNOWN_METHOD_DEFS

// IPageLayoutControlEvents
void OnAfterDraw(VARIANT display, long viewDrawPhase);
void OnAfterScreenDraw(long hdc);
void OnBeforeScreenDraw(long hdc);
void OnDoubleClick(long button, long shift, long x, long y,
                   double mapX, double mapY);
void OnExtentUpdated(VARIANT displayTransformation,
                     VARIANT_BOOL sizeChanged, VARIANT newEnvelope);
void OnFullExtentUpdated(VARIANT displayTransformation,
                         VARIANT newEnvelope);
void OnKeyDown(long keyCode, long shift);
void OnKeyUp(long keyCode, long shift);
void OnFocusMapChanged();
void OnPageLayoutReplaced(VARIANT newPageLayout);
void OnPageSizeChanged();
void OnMouseDown(long button, long shift, long x, long y,
                 double mapX, double mapY);
void OnMouseMove(long button, long shift, long x, long y,
                 double mapX, double mapY);
void OnMouseUp(long button, long shift, long x, long y,
               double mapX, double mapY);
void OnOleDrop(esriControlsDropAction dropAction, VARIANT
              dataObjectHelper, long* effect, long button, long shift,
              long x, long y);
void OnSelectionChanged();
void OnViewRefreshed(VARIANT ActiveView, long viewDrawPhase,
                     VARIANT layerOrElement, VARIANT envelope);
};

#endif // __PAGELAYOUTCONTROLEVENTS_H_
```

3. However, you need to have access to the global PageLayoutControl and MapControl from MapViewer.cpp. You will define them as extern after the #include lines. This tells the compiler that they are defined in another file. This will be accomplished by placing the following code in PageLayoutControlEvents.h:

```
#define __PAGELAYOUTCONTROLEVENTS_H_
```

```
#include <Ao/AoControls.h>

extern IPageLayoutControlPtr g_ipPageLayoutControl;
extern IMapControl3Ptr g_ipMapControl;
```

4. Place the following implementation for IPageLayoutControlEventsHelper's functions into PageLayoutControlEvents.cpp, another new file. Since they are void functions, they can be left empty. Implementation for some of them will be done later in this scenario.

```
#include "PageLayoutControlEvents.h"

void PageLayoutControlEvents::OnAfterDraw(VARIANT display, long
                                          viewDrawPhase)
{
}

void PageLayoutControlEvents::OnAfterScreenDraw(long hdc)
{
}

void PageLayoutControlEvents::OnBeforeScreenDraw(long hdc)
{
}

void PageLayoutControlEvents::OnDoubleClick(long button, long shift, long x,
                                        long y, double mapX, double mapY)
{
}

void PageLayoutControlEvents::OnExtentUpdated(VARIANT displayTransformation,
                                          VARIANT_BOOL sizeChanged,
                                          VARIANT newEnvelope)
{
}

void PageLayoutControlEvents::OnFullExtentUpdated(VARIANT
                                            displayTransformation,
                                            VARIANT newEnvelope)
{
}

void PageLayoutControlEvents::OnKeyDown(long keyCode, long shift)
{
}

void PageLayoutControlEvents::OnKeyUp(long keyCode, long shift)
{
}

void PageLayoutControlEvents::OnFocusMapChanged()
```

```
{
}

void PageLayoutControlEvents::OnPageLayoutReplaced(VARIANT newPageLayout)
{
}

void PageLayoutControlEvents::OnPageSizeChanged()
{
}

void PageLayoutControlEvents::OnMouseDown(long button, long shift, long x,
                                          long y, double mapX, double mapY)
{
}

void PageLayoutControlEvents::OnMouseMove(long button, long shift, long x,
                                          long y, double mapX, double mapY)
{
}

void PageLayoutControlEvents::OnMouseUp(long button, long shift, long x,
                                        long y, double mapX, double mapY)
{
}

void PageLayoutControlEvents::OnOleDrop(esriControlsDropAction dropAction,
                                        VARIANT dataObjectHelper,
                                        long* effect, long button,
                                        long shift, long x, long y)
{
}

void PageLayoutControlEvents::OnSelectionChanged()
{
}

void PageLayoutControlEvents::OnViewRefreshed(VARIANT ActiveView,
                                              long viewDrawPhase,
                                              VARIANT layerOrElement,
                                              VARIANT envelope)
{
}
```

Although all of the functions are left empty, they must all be defined here to prevent the class from being an abstract class.

5. Now you can do the actual update of the MapControl's ActiveView. Enter the following code into the *OnPageLayoutReplaced* event of the PageLayoutControl, which is called whenever a document is loaded into the PageLayoutControl. It will be found in PageLayoutControlEvents.cpp, which you created in the last step.

```
void PageLayoutControlEvents::OnPageLayoutReplaced(VARIANT newPageLayout)
{
```

```
  // Load the same pre-authored map document into the MapControl
  CComBSTR DocFileName;
  IPageLayoutControl2Ptr ipPage2 = g_ipPageLayoutControl;
  ipPage2->get_DocumentFilename(&DocFileName);
  g_ipMapControl->LoadMxFile(DocFileName);
}
```

6. Since events have been added, you need to tell the main application to listen for them. Do this by creating an instance of the new class in the *main* function of MapViewer.cpp and by using the *IEventListenerHelper* interface.

   a. First, include PageLayoutControlEvents.h at the top of MapViewer.h.

```
#include <Ao/AoMotifControls.h>

// Events
#include "PageLayoutControlEvents.h"
```

   b. In MapViewer.cpp, declare the global variables to use in listening for events.

```
IPageLayoutControlPtr g_ipPageLayoutControl;

// Events
PageLayoutControlEvents* g_pageLayoutEvents;
IEventListenerHelperPtr g_ipPageLayoutControlEventHelper;
```

   c. Place the code to begin listening after the *FormSetup* in *main*, and before the data is loaded.

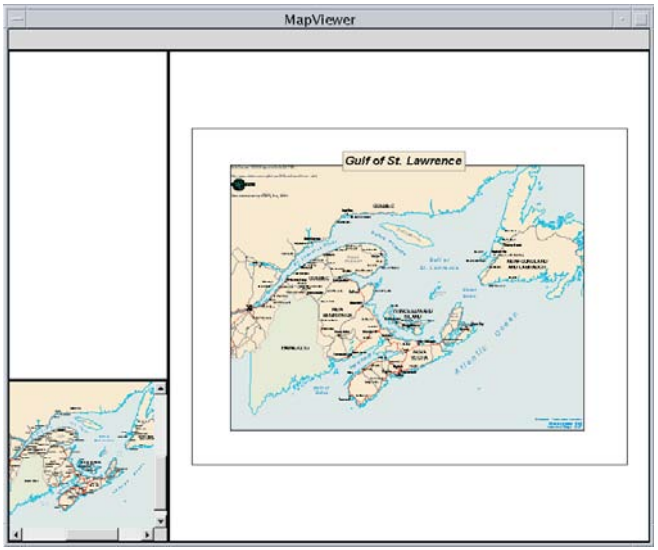
```
FormSetup(argc, argv, &app_context);

// Event listening
g_pageLayoutEvents = new PageLayoutControlEvents();
g_ipPageLayoutControlEventHelper.CreateInstance(
  CLSID_PageLayoutControlEventsListener);
g_ipPageLayoutControlEventHelper->Startup(
  static_cast<IPageLayoutControlEventsHelper*>(g_pageLayoutEvents));
g_ipPageLayoutControlEventHelper->AdviseEvents(g_ipPageLayoutControl,
                                               NULL);
```

7. Before closing the application, you must clean up the events by calling *UnadviseEvents* and *Shutdown*, as well as deleting the instance of *PageLayoutControlEvents*. This is done in *CloseAppCallback* before the control interface pointers are set to 0.

In GTK, the event cleanup will be done in `delete_event`*, a callback used with* `destroy_event` *to close the application.*

```
// Function called when WM_DELETE_WINDOW protocol is passed
void CloseAppCallback(Widget w, XtPointer client_data, XtPointer call_data)
{
  // End event listening
  g_ipPageLayoutControlEventHelper->UnadviseEvents();
  g_ipPageLayoutControlEventHelper->Shutdown();
  g_ipPageLayoutControlEventHelper = 0;
  delete g_pageLayoutEvents;

g_ipToolbarControl = 0;
```

8. Update the makefile. Include PageLayoutControlEvents.cpp as a source, add the header for the events to the MapViewer.o dependencies list, and add a dependencies list for PageLayoutControlEvents.o.
9. Compile and run the application. The map document is now loaded into the PageLayoutControl; and the TOCControl lists the data layers in the MapDocument. By default, the focus map of the map document is loaded into the MapControl.



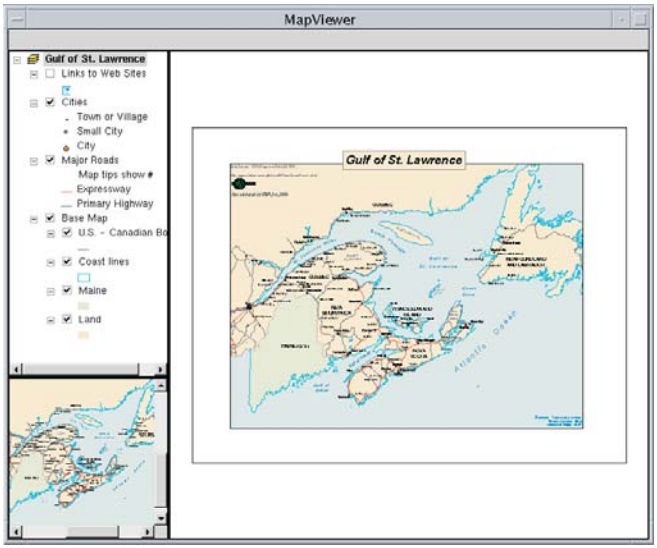
Setting the Buddy Control for the TOCControl and ToolbarControl

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. The buddy control is set after the widgets are created and their interface pointers have been assigned, so you will set each one after FormSetup().

```
FormSetup(argc, argv, &app_context);

// Buddy the toolbar and TOC with the PageLayoutControl
g_ipToolbarControl->SetBuddyControl(g_ipPageLayoutControl);
g_ipTOCControl->SetBuddyControl(g_ipPageLayoutControl);

// Event listening
```

2. Remake the application and run it again. Notice that now the TOCControl displays a layer icon as well as the data for the current document. Use the TOCControl to toggle layer visibility by checking and unchecking the boxes. At this point the ToolbarControl is empty because no commands have been added to it.



Adding commands to the ToolbarControl

ArcGIS Engine also provides commands for use with the SceneControl and GlobeControl.

ArcGIS Engine comes with more than 120 commands and tools that work with the MapControl, PageLayoutControl, and 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. Create a new function to add commands and tools to the ToolbarControl.

   a. The forward declaration of the function follows that for LoadData() in MapViewer.h:

   ```
   void LoadData();
   void AddToolbarItems();
   ```

   b. The function implementation for adding the prebuilt commands and tools goes in MapViewer.cpp after the *LoadData* function.

   ```
   void LoadData()
   {
   …
   }

   void AddToolbarItems()
   {
    long itemIndex;
    CComVariant varTool;

    varTool = L"esriControlCommands.ControlsOpenDocCommand";
   ```

```
g_ipToolbarControl->AddItem(varTool, 0, -1, VARIANT_FALSE, 0,
                            esriCommandStyleIconOnly, &itemIndex);

// Add PageLayout Navigation Commands
varTool = L"esriControlCommands.ControlsPageZoomInTool";
g_ipToolbarControl->AddItem(varTool, 0, -1, VARIANT_TRUE, 0,
                            esriCommandStyleIconOnly, &itemIndex);
varTool = L"esriControlCommands.ControlsPageZoomOutTool";
g_ipToolbarControl->AddItem(varTool, 0, -1, VARIANT_FALSE, 0,
                           esriCommandStyleIconOnly, &itemIndex);
varTool = L"esriControlCommands.ControlsPagePanTool";
g_ipToolbarControl->AddItem(varTool, 0, -1, VARIANT_FALSE, 0,
                            esriCommandStyleIconOnly, &itemIndex);
varTool = L"esriControlCommands.ControlsPageZoomWholePageCommand";
g_ipToolbarControl->AddItem(varTool, 0, -1, VARIANT_FALSE, 0,
                            esriCommandStyleIconOnly, &itemIndex);
 varTool =
L"esriControlCommands.ControlsPageZoomPageToLastExtentBackCommand";
g_ipToolbarControl->AddItem(varTool, 0, -1, VARIANT_FALSE, 0,
                            esriCommandStyleIconOnly, &itemIndex);
 varTool =
L"esriControlCommands.ControlsPageZoomPageToLastExtentForwardCommand";
g_ipToolbarControl->AddItem(varTool, 0, -1, VARIANT_FALSE, 0,
                            esriCommandStyleIconOnly, &itemIndex);

// Add Map Navigation Commands
varTool = L"esriControlCommands.ControlsMapZoomInTool";
g_ipToolbarControl->AddItem(varTool, 0, -1, VARIANT_TRUE, 0,
                            esriCommandStyleIconOnly, &itemIndex);
varTool = L"esriControlCommands.ControlsMapZoomOutTool";
g_ipToolbarControl->AddItem(varTool, 0, -1, VARIANT_FALSE, 0,
                            esriCommandStyleIconOnly, &itemIndex);
varTool = L"esriControlCommands.ControlsMapPanTool";
g_ipToolbarControl->AddItem(varTool, 0, -1, VARIANT_FALSE, 0,
                            esriCommandStyleIconOnly, &itemIndex);
varTool = L"esriControlCommands.ControlsMapFullExtentCommand";
g_ipToolbarControl->AddItem(varTool, 0, -1, VARIANT_FALSE, 0,
                            esriCommandStyleIconOnly, &itemIndex);
}
```

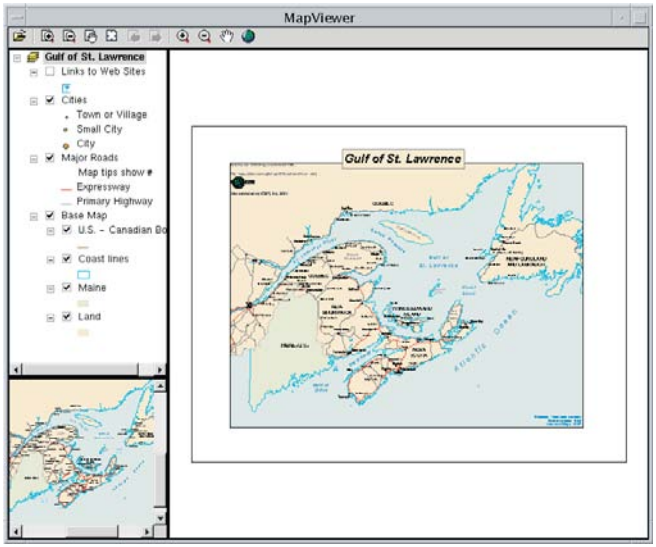
c. Call the new function from *main* after the controls are buddied.

```
g_ipTOCControl->SetBuddyControl(g_ipPageLayoutControl);

AddToolbarItems();
```

2. Remake and 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 page layout and the map commands to navigate around the data present in the data frames. Use the open document command (all the way to the left) to browse to and load other map documents. Notice that not all

Information on the ArcGIS Engine control commands, including GUIDS, descriptions, and which controls each command can interact with, can be found in the ArcGIS Engine Developer Kit Help system under Technical Documents, Names and Ids, Control Commands.

of the tools are initially enabled. If there is no previous or next extent, you cannot zoom to it, and if there is no data, there are no features to select.



Creating a popup menu for the MapControl

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 clicked in the display area of the PageLayoutControl.

1. To implement the popup menu, you will use the *IToolbarMenu* interface. Define the popup menu after the events in MapViewer.cpp.

```
IEventListenerHelperPtr g_ipPageLayoutControlEventHelper;

IToolbarMenuPtr g_ipPopupMenu;
```

2. Create an instance of the popup menu, attaching it to the PageLayoutControl. This is done in *main*.

Instead of using the IToolbarMenu *interface, this could also be done using a Motif popup menu.*

```
g_ipTOCControl->SetBuddyControl(g_ipPageLayoutControl);

// Associate the popup menu with the PageLayoutControl
g_ipPopupMenu.CreateInstance(CLSID_ToolbarMenu);
g_ipPopupMenu->SetHook(g_ipPageLayoutControl);

AddToolbarItems();
```

3. Remember to clean up *g_ipPopupMenu* when the application closes.

```
// Function called when WM_DELETE_WINDOW protocol is passed
void CloseAppCallback(Widget w, XtPointer client_data, XtPointer
call_data)
 {
```

```
// End event listening
g_ipPageLayoutControlEventHelper->UnadviseEvents();
g_ipPageLayoutControlEventHelper->Shutdown();
g_ipPageLayoutControlEventHelper = 0;
delete g_pageLayoutEvents;

g_ipPopupMenu = 0;

g_ipToolbarControl = 0;
```

4. Place some commands on the popup menu. This is done in the *AddPopupItems* function.

   a. Again, start with a forward declaration in the header file, MapViewer.h:

```
void AddToolbarItems();
void AddPopupItems();
```

   b. Implement the function at the bottom of MapViewer.cpp.

```
void AddToolbarItems()
{
  …
}

void AddPopupItems()
{
 CComVariant varTool;
 long popupItemIndex;

 varTool = L"esriControlCommands.ControlsPageZoomInFixedCommand";
 g_ipPopupMenu->AddItem(varTool, 0, -1, VARIANT_FALSE,
                        esriCommandStyleIconAndText, &popupItemIndex);
 varTool = L"esriControlCommands.ControlsPageZoomOutFixedCommand";
 g_ipPopupMenu->AddItem(varTool, 0, -1, VARIANT_FALSE,
                        esriCommandStyleIconAndText, &popupItemIndex);
 varTool = L"esriControlCommands.ControlsPageZoomWholePageCommand";
 g_ipPopupMenu->AddItem(varTool, 0, -1, VARIANT_FALSE,
                        esriCommandStyleIconAndText, &popupItemIndex);
 varTool =
L"esriControlCommands.ControlsPageZoomPageToLastExtentBackCommand";
 g_ipPopupMenu->AddItem(varTool, 0, -1, VARIANT_TRUE,
                       esriCommandStyleIconAndText, &popupItemIndex);
   varTool =
 L"esriControlCommands.ControlsPageZoomPageToLastExtentForwardCommand";
  g_ipPopupMenu->AddItem(varTool, 0, -1, VARIANT_FALSE,
                        esriCommandStyleIconAndText, &popupItemIndex);
}
```

   c. Add commands to the popup menu by calling the function just written.

```
g_ipPopupMenu->SetHook(g_ipPageControl);

AddPopupItems();
AddToolbarItems();
```

Note that only tools and commands that are registered on the system as COM components can be added to the popup menu using the AddItem method. Custom C++ commands and tools, like the one you will build later, cannot be added to the popup menu, as they are not registered as COM components in the system registry.

4. To display the popup menu by right-clicking, you will use the *PageLayoutControlEvents* class you created earlier.

   a. First you need to provide access to the global *ToolbarMenu* defined in MapViewer.cpp. Update PageLayoutControlEvents.h with the following code:

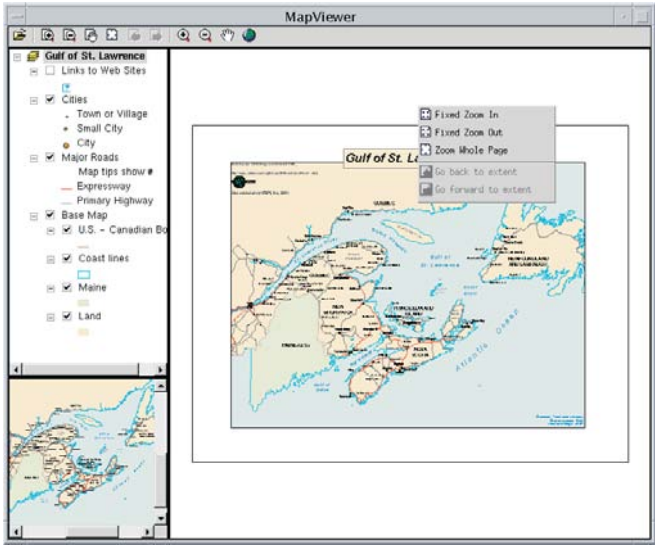
```
extern IMapControl3Ptr g_ipMapControl;
extern IToolbarMenuPtr g_ipPopupMenu;
```

   b. Implement the PageLayoutControlEvent.cpp file's *OnMouseDown* event so that the popup menu displays upon right-clicking in the PageLayoutControl's area.

```
void PageLayoutControlEvents::OnMouseDown(long button, long shift, long x,
                                          long y, double mapX, double map Y)
{
  // Popup the ToolbarMenu
  if (button == 2)
     {
      long lHWndParent;
      g_ipPageLayoutControl->get_hWnd(&lHWndParent);
      g_ipPopupMenu->PopupMenu(x, y, lHWndParent);
     }
}
```

If using a Motif popup menu instead of the IToolbarMenu *interface, this would be done with a Motif event handler. Mixing Motif widget implementations with ArcObjects event handling, or ArcGIS objects with Motif error handling will result in undetermined behavior and is not recommended.*

5. Remake and run the application. Right-click 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. The TOCControl label editing events must be triggered. To do so, add the following code to MapViewer.cpp in *main* after *FormSetup*.

```
FormSetup(argc, argv, &app_context);

g_ipTOCControl->put_LabelEdit(esriTOCControlManual);
```

2. You must listen for events in the TOCControl by writing a class that inherits from *ITOCControlEventsHelper*, as you did for the PageLayoutControl earlier. Start by creating TOCControlEvents.h, a new text file:

```
#ifndef __TOCCONTROLEVENTS_H_
#define __TOCCONTROLEVENTS_H_

#include <ArcSDK.h>
#include <Ao/AoControls.tlh>

class TOCControlEvents : public ITOCControlEventsHelper
{

public:

// IUnknown
IUNKNOWN_METHOD_DEFS

// ITOCControlEvents
void OnMouseDown(long button, long shift, long x, long y);
void OnMouseUp(long button, long shift, long x, long y);
void OnMouseMove (long button, long shift, long x, long y);
void OnDoubleClick (long button, long shift, long x, long y);
void OnKeyDown (long keyCode, long shift);
void OnKeyUp (long keyCode, long shift);
void OnBeginLabelEdit (long x, long y, VARIANT_BOOL* CanEdit);
void OnEndLabelEdit (long x, long y, BSTR newLabel, VARIANT_BOOL*
                     CanEdit);
};

#endif // __TOCCONTROLEVENTS_H_
```

3. Place the implementation for *ITOCControlEventsHelper*'s functions into TOCControlEvents.cpp, another new file. Since they are void functions, they can be left empty. However, you will implement *OnEndLabelEdit*. In that function, tell the TOC to forbid the edit if the new label is an empty string.

```
#include "TOCControlEvents.h"

void TOCControlEvents::OnMouseDown(long button, long shift, long x, long y)
{
```

```
}

void TOCControlEvents::OnMouseUp(long button, long shift, long x, long y)
{
}

void TOCControlEvents::OnMouseMove (long button, long shift, long x, long y)
{
}

void TOCControlEvents::OnDoubleClick (long button, long shift, long x, long y)
{
}

void TOCControlEvents::OnKeyDown (long keyCode, long shift)
{
}

void TOCControlEvents::OnKeyUp (long keyCode, long shift)
{
}

void TOCControlEvents::OnBeginLabelEdit (long x, long y,
                                         VARIANT_BOOL* CanEdit)
{
}

void TOCControlEvents::OnEndLabelEdit (long x, long y, BSTR newLabel,
                                       VARIANT_BOOL* CanEdit)
{
  if (CComBSTR("") == newLabel)
     *CanEdit = VARIANT_FALSE;
}
```

4. Now that the events have been implemented, the main application can listen for them. This is done the same way it was done for the PageLayoutControl's events.

   a. First include TOCControlEvents.h in MapViewer.h.

   ```
   #include "PageLayoutControlEvents.h"
   #include "TOCControlEvents.h"
   ```

   b. Declare some global variables for the TOCControl events at the top of MapViewer.cpp.

   ```
   IEventListenerHelperPtr g_ipPageLayoutControlEventHelper;
   TOCControlEvents* g_tocEvents;
   IEventListenerHelperPtr g_ipTOCControlEventHelper;
   ```

   c. Next place the code to start listening for the *TOCControlEvents* after that for the *PageLayoutControlEvents* in MapViewer.cpp's *main*.

   ```
   g_ipPageLayoutControlEventHelper->AdviseEvents(g_ipPageLayout, NULL);

   g_tocEvents = new TOCControlEvents();
   g_ipTOCControlEventHelper.CreateInstance(CLSID_TOCControlEventsListener);
   ```

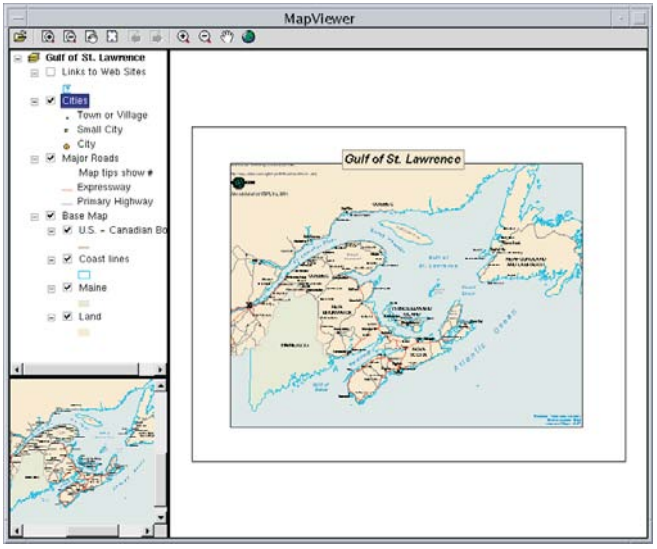
```
    g_ipTOCControlEventHelper->Startup(
        static_cast<ITOCControlEventsHelper*> (g_tocEvents));
    g_ipTOCControlEventHelper->AdviseEvents(g_ipTOCControl, NULL);
```

5. Don't forget to clean up the TOC's events in *CloseAppCallback*. This is done the same as it was for the *PageLayout*'s events.

```
// End event listening
g_ipPageLayoutControlEventHelper->UnadviseEvents();
g_ipPageLayoutControlEventHelper->Shutdown();
g_ipPageLayoutControlEventHelper = 0;
delete g_pageLayoutEvents;
g_ipTOCControlEventHelper->UnadviseEvents();
g_ipTOCControlEventHelper->Shutdown();
g_ipTOCControlEventHelper = 0;
delete g_tocEvents;
```

6. Update the makefile. Include TOCControlEvents.cpp as a source, add the header of the events to the MapViewer.o dependencies list, and add a dependencies list for TOCControlEvents.o.

7. Compile and run the application. To edit a map, layer, heading, or legend class label in the TOCControl, click it once, then click 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 an overview rectangle 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. First add *IFillSymbol* and *IEnvelope* interface pointers to the top of MapViewer.cpp.

```
IToolbarMenuPtr g_ipPopupMenu;
IFillSymbolPtr g_ipFillSymbol;
IEnvelopePtr g_ipCurrentExtent;

int main(int argc, char* argv[])
```

2. You will use a new function to create the rectangle used on the MapControl to highlight the current PageLayoutControl extent.

   a. First place the forward declaration in MapViewer.h.

```
void AddPopupItems();
HRESULT CreateOverviewSymbol();
```

   b. Place the implementation of that function at the end of MapViewer.cpp.

```
void AddPopupItems()
{
  …
 }

HRESULT CreateOverviewSymbol()
 {
  // IRgbColor interface
  IRgbColorPtr ipColor(CLSID_RgbColor);
  ipColor->put_Red(255);
  ipColor->put_Green(0);
  ipColor->put_Blue(0);
```

Alternatively, symbols can be retrieved from style galleries. When working with style galleries and the C++ API, ServerStyleGallery should be used as it is across platforms.

```
  // ILine symbol interface
  ILineSymbolPtr ipOutline(CLSID_SimpleLineSymbol);
  ipOutline->put_Width(2);
  ipOutline->put_Color(ipColor);

  // IFillSymbol properties
  g_ipFillSymbol.CreateInstance(CLSID_SimpleFillSymbol);
  g_ipFillSymbol->put_Outline(ipOutline);
  ((ISimpleFillSymbolPtr) g_ipFillSymbol)->put_Style(esriSFSHollow);

  return S_OK;
 }
```

   c. Create the symbol in the *main* function of MapViewer.cpp before the TOCControl label editing code.

```
CreateOverviewSymbol();

g_ipTOCControl->put_LabelEdit(esriTOCControlManual);
```

d. Remember to clean up the global variables when the application closes.

```
// Function called when WM_DELETE_WINDOW protocol is passed
void CloseAppCallback(Widget w, XtPointer client_data, XtPointer
call_data)
{
   …
  delete g_tocEvents;

  g_ipPopupMenu = 0;
  g_ipFillSymbol = 0;
  g_ipCurrentExtent = 0;

  g_ipToolbarControl = 0;
```

3. To have the MapControl work as an overview window, it must display the full extent of the data. Since this needs to be done every time there is a new map document, the following code should be placed into the PageLayoutControlEvents.cpp file's *OnPageLayoutReplaced* event.

```
void PageLayoutControlEvents::OnPageLayoutReplaced(VARIANT newPageLayout)

{
  // Load the same pre-authored map document into the MapControl
  CComBSTR DocFileName;
  IPageLayoutControl2Ptr ipPage2 = g_ipPageLayoutControl;
  ipPage2->get_DocumentFilename(&DocFileName);
  g_ipMapControl->LoadMxFile(DocFileName);

  // Set the extent of the MapControl to be the full extent
  IEnvelopePtr ipFullExtentEnv;
  g_ipMapControl->get_FullExtent(&ipFullExtentEnv);
  g_ipMapControl->put_Extent(ipFullExtentEnv);
}
```

4. The global variable *g_ipCurrentExtent*, which will be used to draw the overview on the MapControl, needs to be updated with every new *PageLayout*. To implement this, you will need access to the *g_ipCurrentExtent* global variable from within the *PageLayoutControlEvents* class. Add the following into PageLayoutControlEvents.h:

```
extern IToolbarMenuPtr g_ipPopupMenu;
extern IEnvelopePtr g_ipCurrentExtent;
```

5. To update the extent rectangle on the overview map to match the extent shown in every new *PageLayout*, you need to set the current extent rectangle to match the visible extent of the *PageLayout*'s map. Do this in the *OnPageLayoutReplaced* event (in PageLayoutControlEvents.cpp).

```
void PageLayoutControlEvents::OnPageLayoutReplaced(VARIANT newPageLayout)

{
  // Get the extent of the PageLayout's focus map
  IActiveViewPtr ipActiveView;
  g_ipPageLayoutControl->get_ActiveView(&ipActiveView);
  IMapPtr ipFocusMap;
```

```
  ipActiveView->get_FocusMap(&ipFocusMap);
  IActiveViewPtr ipMapActiveView(ipFocusMap);
  ipMapActiveView->get_Extent(&g_ipCurrentExtent);

  // Load the same pre-authored map document into the MapControl
  …
}
```

6. The *PageLayoutControlEvents* do not indicate when the extent of the map within the data frame changes. To receive that information you will use the *ITransformEvents* interface of the PageLayoutControl's focus map. Implementing a class, *TransformEvents*, that extends *ITransformEvents* accomplishes this. It will need to update the extent envelope and refresh the MapControl in *VisibleExtentUpdated*. To do so, your new class will need access to the *g_ipMapControl* and *g_ipCurrentExtent* global variables. Create a new file, TransformEvents.h, with the following code:

```
#ifndef __TRANSFORMEVENTS_H_
#define __TRANSFORMEVENTS_H_

// ArcObjects Headers
// Engine
#include <ArcSDK.h>
// Controls
#include <Ao/AoControls.h>

extern IMapControl3Ptr g_ipMapControl;
extern IEnvelopePtr g_ipCurrentExtent;

class TransformEvents : public ITransformEvents
{
  public:

  // IUnknown
  IUNKNOWN_METHOD_DEFS

  // ITransformEvents
  HRESULT BoundsUpdated(IDisplayTransformation* sender);
  HRESULT DeviceFrameUpdated(IDisplayTransformation* sender,
                             VARIANT_BOOL sizeChanged);
  HRESULT ResolutionUpdated(IDisplayTransformation* sender);
  HRESULT RotationUpdated(IDisplayTransformation* sender);
  HRESULT UnitsUpdated(IDisplayTransformation* sender);
  HRESULT VisibleBoundsUpdated(IDisplayTransformation* sender,
                               VARIANT_BOOL sizeChanged);
};

#endif // __TRANSFORMEVENTS_H_
```

7. Implement that class by placing the following code in TransformEvents.cpp, another new file. In particular, pay attention to *VisibleBoundsUpdated*. 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.

```
#include "TransformEvents.h"

HRESULT TransformEvents::BoundsUpdated(IDisplayTransformation* sender)
{
  return E_NOTIMPL;
}

HRESULT TransformEvents::DeviceFrameUpdated(IDisplayTransformation* sender,
                                            VARIANT_BOOL sizeChanged)
{
  return E_NOTIMPL;
}

HRESULT TransformEvents::ResolutionUpdated(IDisplayTransformation* sender)
{
  return E_NOTIMPL;
}

HRESULT TransformEvents::RotationUpdated(IDisplayTransformation* sender)
{
  return E_NOTIMPL;
}

HRESULT TransformEvents::UnitsUpdated(IDisplayTransformation* sender)
{
  return E_NOTIMPL;
}

HRESULT TransformEvents::VisibleBoundsUpdated(IDisplayTransformation* sender,
                                              VARIANT_BOOL sizeChanged)
{
  // Set the extent to the new visible extent
  sender->get_VisibleBounds(&g_ipCurrentExtent);

  // Refresh the MapControl's foreground phase
  HRESULT hr = g_ipMapControl->Refresh(esriViewForeground);
  return hr;
}
```

8. Although the *TransformEvents* class has been implemented, those events are not yet listened for.

   a. First, include the new TransformEvents.h header file in MapViewer.h:

   ```
   #include "TOCControlEvents.h"
   #include "TransformEvents.h"
   ```

   b. Next start up these events in MapViewer.cpp's *main*, but you will not advise them there. Remember to place the variable declarations at the top of MapViewer.cpp.

   ```
   TOCControlEvents* g_tocEvents;
   ```

```
IEventListenerHelperPtr g_ipTOCControlEventHelper;
TransformEvents* g_transEvents;
IEventListenerHelperPtr g_ipTransEventHelper;

…
int main (int argc, char* argv[])
{
  …
 g_ipTOCControlEventHelper->AdviseEvents(g_ipTOCControl, NULL);

 g_transEvents = new TransformEvents();
 g_ipTransEventHelper.CreateInstance(CLSID_TransformEventsListener);
 g_ipTransEventHelper->Startup(
  static_cast<TransformEvents*> (g_transEvents));

  …
}
```

9. You need to trap for the *TransformEvents* from the *PageLayoutControlEvents* *OnPageLayoutReplaced* event.

   a. To do this *PageLayoutControlEvents* will need to know about *g_ipTransEventHelper*, so declare it with extern in PageLayoutControlEvents.h.

```
extern IEnvelopePtr g_ipCurrentExtent;
extern IEventListenerHelperPtr g_ipTransEventHelper;
```

   b. Now advise the events in *OnPageLayoutReplaced*:

```
void PageLayoutControlEvents::OnPageLayoutReplaced(VARIANT newPageLayout)
{
 // Get the extent of the PageLayout's focus map
 IActiveViewPtr ipActiveView;
 g_ipPageLayoutControl->get_ActiveView(&ipActiveView);
 IMapPtr ipFocusMap;
 ipActiveView->get_FocusMap(&ipFocusMap);
 IActiveViewPtr ipMapActiveView = ipFocusMap;
 ipMapActiveView->get_Extent(&g_ipCurrentExtent);

 // Trap focus map's ITransformEvents
 IScreenDisplayPtr ipScreenDisp;
 ipMapActiveView->get_ScreenDisplay(&ipScreenDisp);
 IDisplayTransformationPtr ipDisplayTrans;
 ipScreenDisp->get_DisplayTransformation(&ipDisplayTrans);
 CComBSTR bsGUID;
 ::StringFromIID(IID_ITransformEvents, &bsGUID);
 IUIDPtr ipUID(CLSID_UID);
 ipUID->put_Value(CComVariant(bsGUID));
 g_ipTransEventHelper->AdviseEvents(ipDisplayTrans, ipUID);

 // Load the same pre-authored map document into the MapControl
  …
}
```

10. Clean up the transform events in MapViewer.cpp's *CloseAppCallback*.

```
// End event listening
g_ipPageLayoutControlEventHelper->UnadviseEvents();
g_ipPageLayoutControlEventHelper->Shutdown();
g_ipPageLayoutControlEventHelper = 0;
delete g_pageLayoutEvents;
g_ipTOCControlEventHelper->UnadviseEvents();
g_ipTOCControlEventHelper->Shutdown();
g_ipTOCControlEventHelper = 0;
delete g_tocEvents;
g_ipTransEventHelper->UnadviseEvents();
g_ipTransEventHelper->Shutdown();
g_ipTransEventHelper = 0;
delete g_transEvents;
```

11. Update the makefile to reflect the new source file. Also, add a dependencies list for TransformEvents.o and add the *TransformEvents* class to the *PageLayoutControlEvent* and *MapViewer* dependencies lists.

12. To do the actual drawing of the symbol on the MapControl, you need to listen for the MapControl's *OnAfterDraw* event. You will implement an event class for the MapControl as you have done the PageLayoutControl and TOCControl. This class will need to know about the global MapControl, extent, and fill symbol. Start with a new file, MapControlEvents.h:

```
#ifndef __MAPCONTROLEVENTS_H_
#define __MAPCONTROLEVENTS_H_

// ArcObjects Headers
// Engine
#include <ArcSDK.h>
// Controls
#include <Ao/AoControls.h>

extern IMapControl3Ptr g_ipMapControl;
extern IEnvelopePtr g_ipCurrentExtent;
extern IFillSymbolPtr g_ipFillSymbol;

class MapControlEvents : public IMapControlEvents2Helper
{
 public:

  // IUnknown
  IUNKNOWN_METHOD_DEFS

  // IMapControlEvents
  void OnAfterDraw(VARIANT display, long viewDrawPhase);
  void OnAfterScreenDraw(long hdc);
  void OnBeforeScreenDraw(long hdc);
  void OnDoubleClick(long button, long shift, long x, long y,
                     double mapX,double mapY);
  void OnExtentUpdated(VARIANT displayTransformation,
```

```
                        VARIANT_BOOL sizeChanged, VARIANT newEnvelope);
  void OnFullExtentUpdated(VARIANT displayTransformation,
                           VARIANT newEnvelope);
  void OnKeyDown(long keyCode, long shift);
  void OnKeyUp(long keyCode, long shift);
  void OnMapReplaced(VARIANT newMap);
  void OnMouseDown(long button, long shift, long x, long y,
                   double mapX, double mapY);
  void OnMouseMove(long button, long shift, long x, long y,
                   double mapX, double mapY);
  void OnMouseUp(long button, long shift, long x, long y,
                 double mapX, double mapY);
  void OnOleDrop(esriControlsDropAction dropAction, VARIANT
                 dataObjectHelper, long* effect, long button, long
                 shift, long x, long y);
  void OnSelectionChanged();
  void OnViewRefreshed(VARIANT ActiveView, long viewDrawPhase,
                       VARIANT layerOrElement, VARIANT envelope);
};

#endif // __MAPCONTROLEVENTS_H_
```

13. Implement those events in MapControlEvents.cpp. Leave all of the functions blank except *OnAfterDraw*, in which the rectangle will be drawn on the MapControl.

```
#include "MapControlEvents.h"

void MapControlEvents::OnAfterDraw(VARIANT display, long viewDrawPhase)
{
  if (g_ipCurrentExtent == 0)
    return;

  // If the foreground phase has drawn, viewDrawPhase will be 32
  esriViewDrawPhase drawPhase = esriViewDrawPhase(viewDrawPhase);
  if (drawPhase == esriViewForeground)
 {
    // Draw the shape on the MapControl
    CComVariant varSymbol = CComVariant(g_ipFillSymbol);
    g_ipMapControl->DrawShape((IGeometryPtr) g_ipCurrentExtent,
                              &varSymbol);
  }
}

void MapControlEvents::OnAfterScreenDraw(long hdc)
{
}

void MapControlEvents::OnBeforeScreenDraw(long hdc)
{
}
```

```
void MapControlEvents::OnDoubleClick(long button, long shift, long x,
                                     long y, double mapX, double mapY)
{
}

void MapControlEvents::OnExtentUpdated(VARIANT displayTransformation,
                                       VARIANT_BOOL sizeChanged,
                                       VARIANT newEnvelope)
{
}

void MapControlEvents::OnFullExtentUpdated(VARIANT displayTransformation,
                                           VARIANT newEnvelope)
{
}

void MapControlEvents::OnKeyDown(long keyCode, long shift)
{
}

void MapControlEvents::OnKeyUp(long keyCode, long shift)
{
}

void MapControlEvents::OnMapReplaced(VARIANT newMap)
{
}

void MapControlEvents::OnMouseDown(long button, long shift, long x, long y,
                                   double mapX, double mapY)
{
}

void MapControlEvents::OnMouseMove(long button, long shift, long x, long y,
                                   double mapX, double mapY)
{
}

void MapControlEvents::OnMouseUp(long button, long shift, long x, long y,
                                 double mapX, double mapY)
{
}

void MapControlEvents::OnOleDrop(esriControlsDropAction dropAction,
                                 VARIANT dataObjectHelper, long* effect,
                                 long button, long shift, long x, long y)
{
}

void MapControlEvents::OnSelectionChanged()
```

```
{
}

void MapControlEvents::OnViewRefreshed(VARIANT ActiveView, long viewDrawPhase,
                                       VARIANT layerOrElement,
                                       VARIANT envelope)
{
}
```

14. These events need to be listened for as well.

    a. First, include the necessary header file in MapViewer.h.

    ```
    #include "TransformEvents.h"
    #include "MapControlEvents.h"
    ```

    b. Listen for the events from MapViewer.cpp in the same way you did for the other controls.

    ```
    IEventListenerHelperPtr g_ipTransEventHelper;
    MapControlEvents* g_mapEvents;
    IEventListenerHelperPtr g_ipMapControlEvent2Helper;

      …

    int main (int argc, char* argv[])
     {
     …

    g_ipTransEventHelper->Startup(
       static_cast<TransformEvents*> (g_transEvents));

    g_mapEvents = new MapControlEvents();
    g_ipMapControlEvent2Helper.CreateInstance(
       CLSID_MapControlEvents2Listener);
    g_ipMapControlEvent2Helper->Startup(
       static_cast<IMapControlEvents2Helper*> (g_mapEvents));
    g_ipMapControlEvent2Helper->AdviseEvents(g_ipMapControl, NULL);

      …
    }
    ```

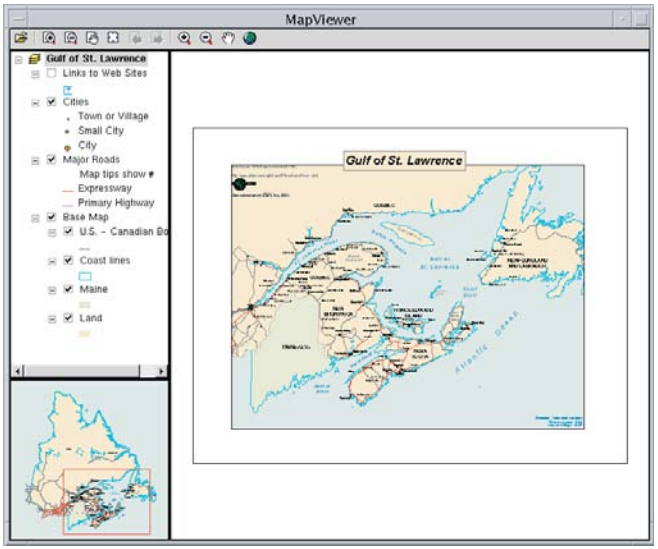
15. Clean up the map's events in *CloseAppCallback* as you did for the other events.

```
// End event listening
g_ipPageLayoutControlEventHelper->UnadviseEvents();
g_ipPageLayoutControlEventHelper->Shutdown();
g_ipPageLayoutControlEventHelper = 0;
delete g_pageLayoutEvents;
g_ipTOCControlEventHelper->UnadviseEvents();
g_ipTOCControlEventHelper->Shutdown();
g_ipTOCControlEventHelper = 0;
delete g_tocEvents;
g_ipTransEventHelper->UnadviseEvents();
g_ipTransEventHelper->Shutdown();
```

```
g_ipTransEventHelper = 0;
delete g_transEvents;
g_ipMapControlEvent2Helper->UnadviseEvents();
g_ipMapControlEvent2Helper->Shutdown();
g_ipMapControlEvent2Helper = 0;
delete g_mapEvents;
```

16. Update the makefile to reflect the MapControlEvents source file. Also, add a dependencies list for MapControlEvents.o and add the *MapControlEvents* class to the MapViewer.o dependencies list.

17. Compile 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 as a red rectangle.



Creating a custom tool

You are not limited to placing the ArcGIS Engine commands and tools on the ToolbarControl. Next, you will create a custom tool that adds to the PageLayoutControl a text element containing today's date at the location of a mouse click. However, this tool will be created as a generic tool so that it could instead 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, copy the AddDate.h and AddDate.cpp files, along with the Res folder from the MapViewer folder, to the directory you are using for this scenario and proceed to Step 5.

1. In your text editor, start a new file, AddDate.h.
2. In AddData.h, create a new class, *AddDateTool*, which inherits from *CAoToolBase*. Include a public constructor and destructor.

```
#ifndef __ADD_DATE_H_
#define __ADD_DATE_H_

// ArcObjects Headers
// Engine
#include <ArcSDK.h>
// Controls
#include <Ao/AoControls.h>
// Custom Tool
#include <Ao/AoToolBase.h>

class AddDateTool : public CAoToolBase
{
 public:
 AddDateTool();
 ~AddDateTool();
};

#endif  // #define __ADD_DATE_H_
```

Since the non-GUI-specific Ao/AoControls.h and Ao/AoToolBase.h are used, the custom tool, like the custom events, will work with both GTK and Motif applications.

3. Since you are writing a new tool, it must implement both *ICommand* and *ITool*, both defined in *CAoToolBase*. In AddDate.h you place the declarations for the functions supported by *ICommand* and *ITool*. For this command to work with all the different controls, you will use the *IHookHelper* interface, storing the hook in a private member variable, m_ipHookHelper. You will also provide member variables for the tool's icon and bitmap.

```
class AddDateTool : public CAoToolBase
{
 public:
 AddDateTool();
 ~AddDateTool();

 // ICommand
 HRESULT get_Enabled(VARIANT_BOOL* Enabled);
 HRESULT get_Checked(VARIANT_BOOL* Checked) ;
 HRESULT get_Name(BSTR* Name);
 HRESULT get_Caption(BSTR* Caption);
 HRESULT get_Tooltip(BSTR* Tooltip);
 HRESULT get_Message(BSTR* Message);
 HRESULT get_Bitmap(OLE_HANDLE* bitmapFile);
 HRESULT get_Category(BSTR* categoryName);
 HRESULT OnCreate(IDispatch* hook);
 HRESULT OnClick();

 // ITool
 HRESULT get_Cursor(OLE_HANDLE* cursorName);
 HRESULT OnMouseDown(LONG Button, LONG Shift, LONG X, LONG Y);
```

```
  HRESULT OnMouseMove(LONG Button, LONG Shift, LONG X, LONG Y);
  HRESULT OnMouseUp(LONG Button, LONG Shift, LONG X, LONG Y);
  HRESULT OnDblClick();
  HRESULT OnKeyDown(LONG keyCode, LONG Shift);
  HRESULT OnKeyUp(LONG keyCode, LONG Shift);
  HRESULT OnContextMenu(LONG X, LONG Y, VARIANT_BOOL* handled);
  HRESULT Refresh(OLE_HANDLE ole);
  HRESULT Deactivate(VARIANT_BOOL* complete);

  private:
  IHookHelperPtr m_ipHookHelper;
  OLE_HANDLE m_hBitmap;
  OLE_HANDLE m_hCursor;
};

#endif // #define FULLEXTENT_H_
```

4. Open a new file in your text editor, and name it AddDate.cpp. Here you will implement your custom tool.

   a. Include AddDate.h.

   ```
   #include "AddDate.h"
   ```

   b. In the constructor, you will load the bitmap and cursor, as well as create the *IHookHelper*. If you want to use the provided icon and mouse cursor, copy the resources from arcgis/developerkit/samples/Developer_Guide_Scenarios/ArcGIS_Engine/Building_an_ArcGIS_Control_Application/Map_Viewer/Res/ to your code directory.

   ```
   #include "AddDate.h"

   AddDateTool::AddDateTool()
    {
      m_ipHookHelper.CreateInstance(CLSID_HookHelper);

      // Load the cursor
      ISystemMouseCursorPtr ipSysMouseCur(CLSID_SystemMouseCursor);
      ipSysMouseCur->LoadFromFile(CComBSTR(L"../Res/date.cur"));
      OLE_HANDLE hTmp;
      HRESULT hr = ipSysMouseCur->get_Cursor(&hTmp);
      if (SUCCEEDED(hr))
       {
         m_hCursor = hTmp;
       }

      // Load the bitmap
      IRasterPicturePtr ipRastPict(CLSID_BasicRasterPicture);
      IPicturePtr ipPict;
      hr = ipRastPict->LoadPicture(CComBSTR(L"../Res/date.bmp"), &ipPict);
      if (SUCCEEDED(hr))
       {
         OLE_HANDLE hBitmap;
   ```

```
    hr = ipPict->get_Handle(&hBitmap);
    if (SUCCEEDED(hr))
     m_hBitmap = hBitmap;
   }
 }
```

c. In the destructor, you will clean up all the interface member variables.

```
AddDateTool::AddDateTool()
{
  …
}

AddDateTool::~AddDateTool()
{
  m_ipHookHelper = 0;
  m_hBitmap = 0;
  m_hCursor = 0;
}
```

d. You now need to stub out all the functions from *ICommand* in AddDate.cpp, even if you are not going to use some of these. Add the following code to the *ICommand* properties and methods:

```
AddDateTool::~AddDateTool()
{
  …
}

HRESULT AddDateTool::get_Enabled(VARIANT_BOOL* Enabled)
{
 if (!Enabled)
   return E_POINTER;

 *Enabled = VARIANT_TRUE;
 return S_OK;
}

HRESULT AddDateTool::get_Checked(VARIANT_BOOL* Checked)
{
 if (!Checked)
   return E_POINTER;

 return S_OK;
}

HRESULT AddDateTool::get_Name(BSTR* Name)
{
 if (!Name)
   return E_POINTER;

 *Name = ::AoAllocBSTR(L"CustomCommands_AddDate");
 return S_OK;
```

```
}

HRESULT AddDateTool::get_Caption(BSTR* Caption)
{
  if (!Caption)
    return E_POINTER;

  *Caption = ::AoAllocBSTR(L"Add Date");
  return S_OK;
}

HRESULT AddDateTool::get_Tooltip(BSTR* Tooltip)
{
  if (!Tooltip)
    return E_POINTER;

  *Tooltip = ::AoAllocBSTR(L"Add date");
  return S_OK;
}

HRESULT AddDateTool::get_Message(BSTR* Message)
{
  if (!Message)
    return E_POINTER;

  *Message = ::AoAllocBSTR(L"Adds a date element to the page layout");
  return S_OK;
}

HRESULT AddDateTool::get_Bitmap(OLE_HANDLE* bitmap)
{
  if (!bitmap)
    return E_POINTER;

  if (m_hBitmap != 0)
   {
    *bitmap = m_hBitmap;
    return S_OK;
   }

  return E_FAIL;
}

HRESULT  AddDateTool::get_Category(BSTR* categoryName)
{
  if (!categoryName)
    return E_POINTER;

  *categoryName = ::AoAllocBSTR(L"CustomCommands");
  return S_OK;
```

```
}

// Create the command and set who it will work with
HRESULT AddDateTool::OnCreate(IDispatch* hook)
{
 if (!hook)
  return E_POINTER;

 m_ipHookHelper->putref_Hook(hook);
 return S_OK;
}

HRESULT AddDateTool::OnClick()
{
 return S_OK;
}
```

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 to 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 a* MapControl, PageLayoutControl, *or* ToolbarControl*).*

e. Write a function that will format the date for display on the PageLayoutControl.

   i. Before the class in AddDate.h, include the following header files:

```
#include <Ao/AoToolBase.h>

#include <time.h>
#include <stdio.h>
```

   ii. Add a private function to the *AddDate* class in AddDate.h to take care of the formatting.

```
OLE_HANDLE m_hCursor;
char* FormatDate();
```

   iii. Implement the function at the bottom of AddDate.cpp.

```
char* AddDateTool::FormatDate()
 {
  time_t dateInfo = time(NULL);
  tm* todaysDate = localtime(&dateInfo);
  int month = todaysDate->tm_mon + 1;
  int day = todaysDate->tm_mday;
  int year = todaysDate->tm_year + 1900;
  char* dateDisplay = new char[12];
  sprintf(dateDisplay, "%d/%d/%d\n", month, day, year);
  return dateDisplay;
 }
```

f. Continue implementing your custom tool by stubbing out all the properties and events of the *ITool* interface before the *FormatDate* function in AddDate.cpp. Pay attention to the implementation of the *OnMouseDown* method, as it creates the date text element and adds it to the graphics container of the application.

```
HRESULT AddDateTool::OnClick()
{
 return S_OK;
}
```

```
HRESULT AddDateTool::get_Cursor(OLE_HANDLE* cursorName)
 {
  if (cursorName == NULL)
    return E_POINTER;

  if (m_hCursor != 0)
   {
    *cursorName = m_hCursor;
    return S_OK;
   }

  return E_FAIL;
 }

// Add the date to the page layout where the mouse is
HRESULT AddDateTool::OnMouseDown(LONG Button, LONG Shift, LONG X, LONG Y)
 {
  if (Button == 1)
   {
    // Format the date & create a text element
    char* dateDisplay = FormatDate();
    ITextElementPtr ipDateTextElem(CLSID_TextElement);
    ipDateTextElem->put_Text(CComBSTR(dateDisplay));
      delete[] dateDisplay;

    ITextSymbolPtr ipDateTextSymb(CLSID_TextSymbol);
    // Add it to the text element
    ipDateTextElem->put_Symbol(ipDateTextSymb);

    // Get point in map display coordinates
    IActiveViewPtr ipActiveView;
    m_ipHookHelper->get_ActiveView(&ipActiveView);
    IScreenDisplayPtr ipScreenDisplay;
    ipActiveView->get_ScreenDisplay(&ipScreenDisplay);
    IDisplayTransformationPtr ipDisplayTrans;
    ipScreenDisplay->get_DisplayTransformation(&ipDisplayTrans);
   IPointPtr ipPoint;
    ipDisplayTrans->ToMapPoint(X, Y, &ipPoint);

    // Set the element's geometry
    ((IElementPtr) ipDateTextElem)->put_Geometry(ipPoint);

    // Add element to the page layout's graphics container
    IGraphicsContainerPtr ipGraphicsContainer;
    ipActiveView->get_GraphicsContainer(&ipGraphicsContainer);
    ipGraphicsContainer->AddElement((IElementPtr) ipDateTextElem, 0);
    ipActiveView->PartialRefresh(esriViewGraphics, NULL, NULL);
  }

    return S_OK;
```

```
}

HRESULT AddDateTool::OnMouseMove(LONG Button, LONG Shift, LONG X, LONG Y)
{
  return E_NOTIMPL;
}

HRESULT AddDateTool::OnMouseUp(LONG Button, LONG Shift, LONG X, LONG Y)
{
  return E_NOTIMPL;
}

HRESULT AddDateTool::OnDblClick()
{
  return E_NOTIMPL;
}

HRESULT AddDateTool::OnKeyDown(LONG keyCode, LONG Shift)
{
  return E_NOTIMPL;
}

HRESULT AddDateTool::OnKeyUp(LONG keyCode, LONG Shift)
{
  return E_NOTIMPL;
}

HRESULT AddDateTool::OnContextMenu(LONG X, LONG Y, VARIANT_BOOL* handled)
{
  return E_NOTIMPL;
}

HRESULT AddDateTool::Refresh(OLE_HANDLE ole)
{
  return E_NOTIMPL;
}

HRESULT AddDateTool::Deactivate(VARIANT_BOOL* complete)
{
  if (!complete)
    return E_POINTER;

  *complete = VARIANT_TRUE;
  return S_OK;
}

char* AddDateTool::FormatDate()
{
   …
}
```

5. Now that you have a working command, it can be incorporated into the application.
   a. Make the application aware of the new command by including AddDate.h in MapViewer.h.

```
#include "MapControlEvents.h"

#include "AddDate.h"
```

   b. Create an instance of the command at the top of MapViewer.cpp.

```
IEnvelopePtr g_ipCurrentExtent;

AddDateTool* g_dateTool;
```

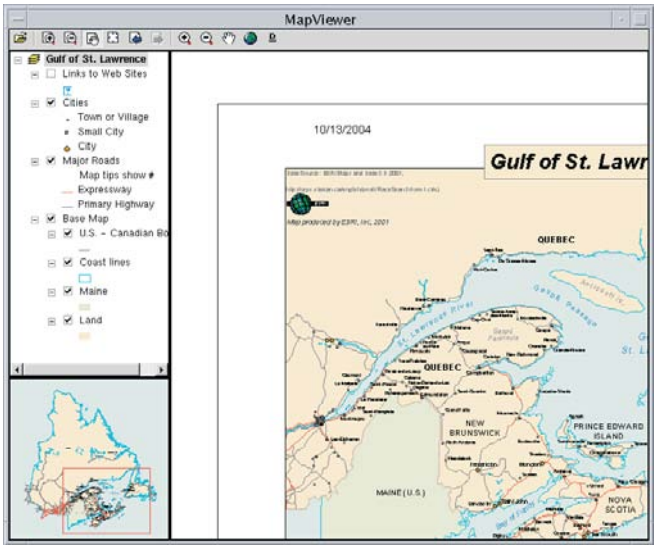
   c. The custom *AddDate* command will be added as the last item on the toolbar. This is done using the *AoToolbarAddTool* C++ API function. Place the call at the end of the *AddToolbarItems* function of MapViewer.cpp.

```
varTool = L"esriControlCommands.ControlsMapFullExtentCommand";
g_ipToolbarControl->AddItem(varTool, 0, -1, VARIANT_FALSE, 0,
                            esriCommandStyleIconOnly, &itemIndex);

// Add custom date placement command to the tools toolbar
g_dateTool = new AddDateTool();
AoToolbarAddTool(g_ipToolbarControl, g_dateTool,
                 esriCommandStyleIconOnly);
```

   d. This instance of the tool must be deleted in *CloseAppCallback*.

```
delete g_mapEvents;

// delete the instance of the tool
delete g_dateTool;
```

6. Update the makefile. List AddDate.cpp as a source and create a dependencies list for AddDate.o. Make sure that the MapViewer.cpp file's dependency on AddDate.h is reflected.
7. Remake and run the application. If you used the provided icon, there will be a new button with a D, underlined twice, on the toolbar. Select your new tool and click the PageLayoutControl to add a text element containing today's date.



Customizing the ToolbarControl

In addition to adding Controls commands to the ToolbarControl programmatically, you can also add them at runtime by customizing the ToolbarControl using the Customize dialog box. To do this, you will place the ToolbarControl in customize mode and display the Customize dialog box.

1. You will place a toggle button next to the ToolbarControl to turn the toolbar customization mode on and off. To follow Motif standards, you will create a new panel for the top of the screen, and it will hold both the ToolbarControl and the toggle button.

   a. First, include Xm/ToggleB.h in MapViewer.h.

   ```
   #include <Xm/Protocols.h>
   #include <Xm/ToggleB.h>
   #undef String
   ```

   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. Custom C++ commands and tools, such as 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.

   b. In MapViewer.cpp, add declarations for the toggle button widget. Also declare the *ICustomizeDialog* interface pointer.

   ```
   IEnvelopePtr g_ipCurrentExtent;
   ICustomizeDialogPtr g_ipCustomizeDialog;

   AddDateTool* g_dateTool;
   Widget g_customizeToggle;
   ```

   c. In *FormSetup*, create the toggle button (after *mainForm* is created) that will allow you to customize the toolbar as well as its panel. Update the widget attachments to reflect the new panel, replacing the location of the ToolbarControl.

   The GTK toggle button's setup is shown in the GTK MapViewer.cpp's `form_setup`.

   ```
   // Create a sub-form to place ToolbarControl and customizeToggle on
   Widget topFormPanel = XtVaCreateWidget("topformpanel",
                                          xmFormWidgetClass,   mainForm,
                                                 XmNtopAttachment,    XmATTACH_FORM,
                                                 XmNrightAttachment,  XmATTACH_FORM,
                                                 XmNleftAttachment,   XmATTACH_FORM,
   ```

```
                                        XmNheight,          25,
                                        NULL);

// customizetoggle setup
XmString label = XmStringCreateLocalized("Customize");
g_customizeToggle = XtVaCreateWidget("customizetoggle",
                              xmToggleButtonWidgetClass, topFormPanel,
                               XmNlabelString,     label,
                              XmNtopAttachment,    XmATTACH_FORM,
                             XmNrightAttachment,  XmATTACH_FORM,
                             XmNbottomAttachment, XmATTACH_FORM,
                              XmNheight,          25,
                              XmNwidth,           150,
                              NULL);
XmStringFree(label);

// ToolbarControl setup
  Widget toolbarWidget = XtVaCreateWidget("toolbarwidget",
                                     mwCtlWidgetClass, mainForm,
                                     mwCtlWidgetClass, topFormPanel,
                                     XmNtopAttachment, XmATTACH_FORM,
                                     XmNleftAttachment, XmATTACH_FORM,
                                     XmNbottomAttachment, XmATTACH_FORM,
                                     XmNrightAttachment, XmATTACH_FORM,
                                     XmNrightAttachment, XmATTACH_WIDGET,
                                     XmNrightWidget, g_customizeToggle,
                                    MwNprogID, AoPROGID_ToolbarControl,
                                     NULL);
XtVaSetValues(toolbarWidget, XmNheight, 25, NULL);
MwCtGetInterface(toolbarwidget, (IUnknown**)&g_ipToolbarControl);

// Create a sub-form to place TOCControl and MapControl on
Widget leftformpanel = XtVaCreateWidget("leftformpanel",
                                     xmFormWidgetClass, mainForm,
                                    XmNtopAttachment, XmATTACH_WIDGET,
                                     XmNtopWidget, toolbarWidget,
                                    XmNtopWidget, topFormPanel,
                                   XmNbottomAttachment, XmATTACH_FORM,
                                    XmNleftAttachment, XmATTACH_FORM,
                                     XmNwidth,           200,
                                     NULL);

...

// PageLayoutControl setup
pagewidget = XtVaCreateWidget("pagewidget",
                               mwCtlWidgetClass,      mainform,
                               XmNtopAttachment,     XmATTACH_WIDGET,
                               XmNtopWidget,          toolbarWidget,
```

```
                                        XmNtopWidget,          topFormPanel,
                                       XmNleftAttachment,    XmATTACH_WIDGET,
                                       XmNleftWidget,         leftFormPanel,
                                       XmNbottomAttachment,    XmATTACH_FORM,
                                       XmNrightAttachment,     XmATTACH_FORM,
                                       MwNprogID,  AoPROGID_PageLayoutControl,
                                        NULL);
    MwCtlGetInterface(pagewidget, (IUnknown**)&g_ipPageLayoutControl);
```

d. Manage the new widgets when the others are managed.

```
XtManageChild(leftFormPanel);
XtManageChild(topFormPanel);
XtManageChild(g_customizeToggle);
XtManageChild(toolbarWidget);
```

2. Add a new function called *CreateCustomizeDialog*. This is where you will create the Customize dialog box.

a. You will place the forward declaration in MapViewer.h.

```
HRESULT CreateOverviewSymbol();
HRESULT CreateCustomizeDialog();
```

b. Define it at the bottom of MapViewer.cpp.

```
HRESULT CreateOverviewSymbol()
{
  …
}

HRESULT CreateCustomizeDialog()
{

 g_ipCustomizeDialog.CreateInstance(CLSID_CustomizeDialog);
 // Set the title
 g_ipCustomizeDialog->put_DialogTitle(
     CComBSTR(L"Customize Toolbar Items"));
 // Don't show the "Add From File" Button
 // Adding from file is not an option for your custom C++ Commands.
 // The C++ API requires programmatic placement of custom commands
 // onto the toolbar control. With the built-in ArcGIS Engine
 // Commands already visible in the dialog, nothing needs to be
 // added from file.
 g_ipCustomizeDialog->put_ShowAddFromFile(VARIANT_FALSE);
 // Set the ToolbarControl that the new items will be added to
 g_ipCustomizeDialog->SetDoubleClickDestination(g_ipToolbarControl);

 return S_OK;
}
```

c. Remember to clean up *g_ipCustomizeDialog* when the application closes.

```
// Function called when WM_DELETE_WINDOW protocol is passed
void CloseAppCallback(Widget w, XtPointer client_data, XtPointer
                      call_data)
{
  …
   delete g_dateTool;

   g_ipPopupMenu = 0;
   g_ipFillSymbol = 0;
   g_ipCurrentExtent = 0;
   g_ipCustomizeDialog = 0;
```

3. Call *CreateCustomizeDialog* from the *main* function sometime after the ToolbarControl is initialized. If this is done any earlier, the Customize dialog box will not be associated with the ToolbarControl. For this scenario, call the function after adding the toolbar items.

```
AddToolbarItems();
CreateCustomizeDialog();
```

4. Create a callback function for the toggle button. When the user clicks the toggle button to the on state, you want to show the Customize dialog box. When it is clicked off, the Customize dialog box should disappear.

If the ArcGIS Engine commands are not appearing in the Customize dialog box, there is a problem with your registry. Those commands can be added from the file, and to have that option in the Customize dialog box, pass VARIANT_TRUE into put_ShowAddFromFile().

a. Place the forward declaration in MapViewer.h.

```
HRESULT CreateCustomizeDialog();
void ToggleCallback(Widget w, XtPointer client_data, XtPointer call_data);
```

b. At the bottom of MapViewer.cpp, place the definition of the callback.

```
HRESULT CreateCustomizeDialog()
{
  …
}

void ToggleCallback(Widget w, XtPointer client_data, XtPointer call_data)
{
 XmToggleButtonCallbackStruct *customize =
   (XmToggleButtonCallbackStruct *) call_data;

 long hWnd;
 g_ipToolbarControl->get_hWnd(&hWnd);

 if (customize->set)
   g_ipCustomizeDialog->StartDialog(hWnd);
 else
   g_ipCustomizeDialog->CloseDialog();
}
```

c. Also set up the callback right after the customizeToggle has been created in *FormSetup*.

```
g_customizeToggle = XtVaCreateWidget(…);
XtAddCallback(g_customizeToggle, XmNvalueChangedCallback, ToggleCallback,
              NULL);
```

5. To put the toolbar into the customize state when the dialog box is started, listen for the *CustomizeDialog* events. Implement a class, *CustomizeDialogEvents*, which inherits from *ICustomizeDialogEvents*.

   a. First make the header file for the class: CustomizeDialogEvents.h. The class will need to have access to the global ToolbarControl and the g_customizeToggle widget. Make sure to include the files needed for the ToolbarControl and the toggle widget. *ICustomizeDialogEvents* is declared with the other classes for the ToolbarControl, so be sure to include toolbarcontrol.tlh and toolbarcontrol_events.tlh.

This class will be Motif specific since it must access the toggle button, a Motif widget. For the GTK-specific code, see CustomizeDialogEvents.h and CustomizeDialogEvents.cpp in the GTK zip file.

```
#ifndef __CUSTOMIZEDIALOGEVENTS_H_
#define __CUSTOMIZEDIALOGEVENTS_H_

// Motif Headers
#define String       esriXString
#define Cursor       esriXCursor
#define Object       esriXObject
#define ObjectClass  esriXObjectClass
#include <Xm/Xm.h>
#include <Xm/MainW.h>
#include <Xm/Form.h>
#include <Xm/Protocols.h>
#include <Xm/ToggleB.h>
#undef String
#undef Cursor
#undef Object
#undef ObjectClass

// ArcObjects Headers
// Engine
#include <ArcSDK.h>
// Controls
#include <Ao/AoControls.h>

extern IToolbarControlPtr g_ipToolbarControl;
extern Widget g_customizeToggle;

class CustomizeDialogEvents : public ICustomizeDialogEvents
{
  public:

    // IUnknown
    IUNKNOWN_METHOD_DEFS

    // ICustomizeDialogEvents
    HRESULT OnStartDialog();
    HRESULT OnCloseDialog();
};

#endif // __CUSTOMIZEDIALOGEVENTS_H_
```

b. Place the implementation for this class into CustomizeDialogEvents.cpp, making sure you include the header file for the class. Implement the functions so that when the dialog box is opened, the toolbar enters the customize state, and when it is closed, the toolbar leaves that state. When it is closed, also make sure to set the toggle button to false, as the dialog box may be closed with a close button that is on it.

```
#include "CustomizeDialogEvents.h"

HRESULT CustomizeDialogEvents::OnStartDialog()
 {
  g_ipToolbarControl->put_Customize(VARIANT_TRUE);

  return S_OK;
 }

HRESULT CustomizeDialogEvents::OnCloseDialog()
 {
  g_ipToolbarControl->put_Customize(VARIANT_FALSE);
  XmToggleButtonSetState(g_customizeToggle, false, true);

  return S_OK;
 }
```

Although the class itself is Motif or GTK specific, listening for either one is done the same way.

c. Like the other event classes, these events must be listened for. Start by including the header file in MapViewer.h.

```
#include "MapControlEvents.h"
#include "CustomizeDialogEvents.h"

#include "AddDate.h"
```

d. Declare variables for those events at the top of MapViewer.cpp.

```
IEventListenerHelperPtr g_ipMapControlEvent2Helper;
CustomizeDialogEvents* g_customizeEvents;
IEventListenerHelperPtr g_ipCustomizeEventHelper;
```

e. Listen for them right after listening for MapControl events.

```
g_ipMapControlEvent2Helper->AdviseEvents(ipMapControl, NULL);

g_customizeEvents = new CustomizeDialogEvents();
g_ipCustomizeEventHelper.CreateInstance(CLSID_CustomizeDialogEventsListener);
g_ipCustomizeEventHelper->Startup(
  static_cast<CustomizeDialogEvents*> (g_customizeEvents));
CComBSTR bsGUID;
::StringFromIID(IID_ICustomizeDialogEvents, &bsGUID);
IUIDPtr ipUID(CLSID_UID);
ipUID->put_Value(CComVariant(bsGUID));
g_ipCustomizeEventHelper->AdviseEvents(g_ipCustomizeDialog, ipUID);
```

f. Clean up the events in *CloseAppCallback*.

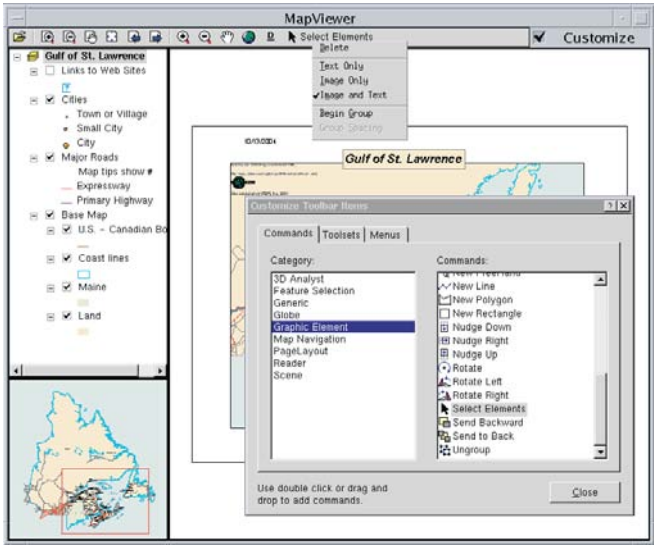
```
// End event listening
g_ipPageLayoutControlEventHelper->UnadviseEvents();
g_ipPageLayoutControlEventHelper->Shutdown();
```

```
g_ipPageLayoutControlEventHelper = 0;
delete g_pageLayoutEvents;
g_ipTOCControlEventHelper->UnadviseEvents();
g_ipTOCControlEventHelper->Shutdown();
g_ipTOCControlEventHelper = 0;
delete g_tocEvents;
g_ipTransEventHelper->UnadviseEvents();
g_ipTransEventHelper->Shutdown();
g_ipTransEventHelper = 0;
delete g_transEvents;
g_ipMapControlEvent2Helper->UnadviseEvents();
g_ipMapControlEvent2Helper->Shutdown();
g_ipMapControlEvent2Helper = 0;
delete g_mapEvents;
g_ipCustomizeEventHelper->UnadviseEvents();
g_ipCustomizeEventHelper->Shutdown();
g_ipCustomizeEventHelper = 0;
delete g_customizeEvents;
```

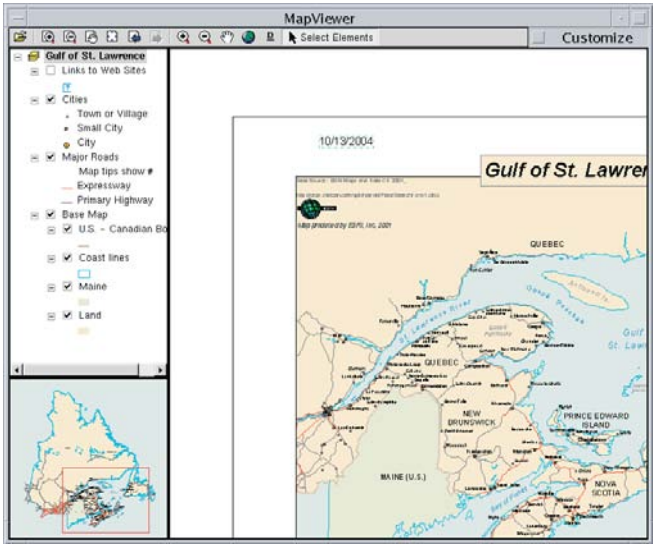
6. Update the makefile. Add CustomizeDialogEvents.cpp to the sources list and add CustomizeDialogEvents.h to the MapViewer.cpp dependencies list. Make a dependencies list for CustomizeDialogEvents.o.

7. Compile and run the application. Check the customize toggle button to put the ToolbarControl into customize mode and open the Customize dialog box.

8. On the Commands tab, choose the Graphic Element category and either drag the Select Elements command to the toolbar or double-click it to add it to the ToolbarControl. By right-clicking an item on the toolbar, you can adjust the appearance in terms of style and grouping. Change the icon you have just added to display both image and text.



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:

- The user machine will require an installation of the ArcGIS Engine Runtime.
- The user's machine will need to have its ArcGIS Engine Runtime initialized.
- The executable created at compile time will need to be deployed onto the user's machine.
- To run: At the command line, type "./MapViewer".

ADDITIONAL RESOURCES

The following resources may help you understand and apply the concepts and techniques presented in this scenario.

- The ArcGIS Engine Developer Kit documentation. This includes component help, object model diagrams, and samples to help you get started.
- The ESRI ArcObjects Online Web site and the ESRI online discussion forums.
- Heller, Dan, Paula M. Ferguson, and David Brennan. *Motif Programming Manual* (The Definitive Guides to the X Window System, Volume 6A) 2nd Edition. O'Reilly & Associates. 1994.
- Oram, Andy, and Steve Talbott. *Managing Projects with make*, 2nd Edition. O'Reilly Press.
- www.gtk.org

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, click Custom or Modify, and click the samples feature under Software Developer Kit.

You can find this sample in:

```
<install_location>\developerkit\samples\Developer_Guide_Scenarios\
Converting_A_Tin_To_Point_ShapefileJava.zip
```

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 a 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* classes 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.

Ant, a cross-platform Java-based build tool, will be used 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.

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, click 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.

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 (including Java) with an authorization file enabling it for development use.

- An installation of the Java 2 Platform, Standard Edition Software Development Kit. See *http://support.esri.com* for information about supported versions of the J2SE SDK. If you do not already have one available, download it from the Java Web site 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.

  `<install_location>\developerKit\samples\DeveloperKit\samples\Developer_Guide_Scenarios\Converting_A_Tin_To_Point_ShapefileJava.zip`

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, click Custom or Modify, and click the samples feature under Software Developer Kit.

Objects from the following packages will be used:

- com.esri.arcgis.datasourcesfile
- com.esri.arcgis.geodatabase
- com.esri.arcgis.geometry
- com.esri.arcgis.system

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 on Windows

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, 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 click 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.

For cross-platform compatibility, the data and pathnames used must be lowercased.

Repeat the steps above to add the following environment variables:

- JAVA_HOME=J2SE SDK install directory

- ANT_HOME=%ARCENGINEHOME\DeveloperKit\tools\ant

Setting up environment variables on Solaris and Linux

To configure your environment on Solaris and Linux you should source two setup scripts. The init_engine script in the ArcGIS installation directory sets the ARCENGINEHOME environment variable as well as several required paths. The setenv_ant script in arcgis/developerkit/tools sets the ANT_HOME variable to point to arcgisant and includes arcgisant's bin directory in your PATH.

- If you use C-shell:

```
source .../arcgis/init_engine.csh
source .../arcgis/developerkit/tools/setenv_ant.csh
```

- If you use bash or bourne shell:

```
source .../arcgis/init_engine.sh
source .../arcgis/developerkit/tools/setenv_ant.sh
```

You should also set the JAVA_HOME environment variable to point to the install location of your J2SE SDK.

To compile and run applications using the ArcGIS Engine Developer Kit, the PATH environment variable should include paths to ArcGIS/bin and J2SE JRE/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.*

- Using C-shell:

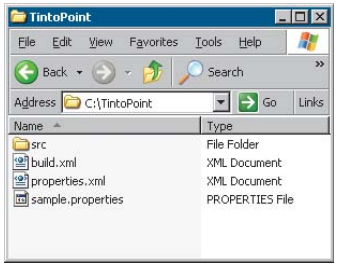
```
setenv JAVA_HOME [J2SE SDK install directory]
```

- Using bash or bourne shell:

```
JAVA_HOME=[J2SE SDK install directory]
```

Build scripts

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 for the scripts to work correctly.



Folder and file structure required for the build scripts to work as desired

- 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 run the scenario.

Creating the sample.properties file

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.

To get an idea of the structure of Ant targets, 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 class path task to ensure the class path settings.

The complete sample target, including closure of the XML tags, 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 have a basic understanding of targets and their usage in Ant, 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, test it before proceeding.

1. Open a command prompt and use the "cd" command to change to 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, you can proceed with writing the Java source code for the example application.

Use your favorite text editor or IDE to write your source code.

Creating the TintoPoint main class

1. Begin your 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.

The datasourcesfile, geodatabase, geometry, and system Java packages are the equivalent of the DataSourcesFile, GeoDatabase, Geometry, and System libraries of ArcGIS Engine.

Next, look at what imports are required by the class.

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 use the *ShapefileWorkspaceFactory* class in this exercise to create your generated 3D shapefile.
- geodatabase—provides all definitions relating to data access including TINs and feature classes. You will be using 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 *AoInitializer* object to initialize and uninitialize 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){
```

The ITinAdvanced *interface requires an ArcGIS Engine Runtime license with 3D extension 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.*

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.

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 the *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 returned by your enumeration. While your node is not null, loop through them and populate your feature class.

```
   // Get tin node enum.
IEnumTinNode nodeEnum = 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 = nodeEnum.IEnumTinNode_next();
```

```
while(node != null){
   node.queryAsPoint(point);
   outBuffer.setShapeByRef(point);
   outCursor.insertFeature(outBuffer);
   node = nodeEnum.IEnumTinNode_next();
}
outCursor.flush();
tinAdv.setEmpty();
```

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 input TIN on disk.

7. Review the code for the full method below to make sure yours 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();
    }

    outCursor.flush();
   tinAdv.setEmpty();
  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 sends 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. Declare a static *AoInitialize* variable and create the signature for this method as follows:

```
private static AoInitialize aoInit;
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{
   aoInit = new AoInitialize();
   aoInit.initialize(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.

This exercise does not implement any logic that determines whether the arguments being passed are valid strings or datapaths.This validation process may be beneficial when developing a production application.

```
/*
 * 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 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 break 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);
try {
   aoInit.shutdown();
} catch (IOException ex) {
   ex.printStackTrace();
}
System.out.println("Tin to Point - Done");
   }
}
```

The code for the command-line application is now complete.

DEPLOYMENT

There are many options for 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 use the "cd" command to change to 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*)—Online version of the book from Addison–Wesley. Learning all about Java.
- Thinking in Java (*http://www.mindview.net/Books/TIJ*)—Online version of the 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 to create a command-line application that 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, click Custom or Modify, and click the samples feature under Software Developer Kit.

You can find this sample in:

```
<install_location>\DeveloperKit\samples\Developer_Guide_Scenarios\
Computing_the_Slope_of_a_Raster_DatasetCpp.zip
```

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 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 extension.
- Performing the calculation of slope on a raster dataset.
- Persisting the resultant raster dataset.
- Deploying the application on all platforms supported by the ArcGIS Engine C++ API.

CONCEPTS

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.

Slope datasets are often used as inputs in physical process models. Slope is commonly expressed in degrees or as a percentage. In the slope calculation a parameter (zFactor) can specify the number of ground x,y units in one z unit. This allows you to create a slope dataset with different z units from 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.

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

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, the same code will work on any other supported platform if built accordingly. VS6 and VS7 .NET 2003 projects, as well as makefiles for both Solaris and Linux, are included with the solution code available in the developer kit.

For more detailed information, see the 'C++ application programming interface' section of Chapter 4, 'Developer environments'.

For cross-platform compatibility, the data and pathnames used must be lowercased.

In the design, some safeguards were taken to ensure that the application remained cross-platform. They include avoiding function calls and data types 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

To successfully follow this scenario you need the following:

The C++ API is not included in the typical installation on Windows of the ArcGIS Engine Developer Kit. If you do not have it installed, rerun the Developer Kit Install wizard, click 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 and additional supported compilers, see the C++ API section of Chapter 4, 'Developer environments', or the online Help.
- A configured ArcObjects environment. For setup details, see the C++ API section of Chapter 4, 'Developer environments', or the online Help.
- 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.

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, click Custom or Modify, and click the samples feature under Software Developer Kit.

- Access to the solution code and makefiles that come with this scenario. This is located at:

  ```
  <install_location>\DeveloperKit\samples\Developer_Guide_Scenarios\
  Computing_the_Slope_of_a_Raster_DatasetCpp.zip
  ```

- An ArcGIS Spatial Analyst or ArcGIS Engine Runtime with Spatial extension license is required for the application to run once deployed.
- To run the application, you will need a raster dataset.

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 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.

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.

This scenario's sample application demonstrates ArcGIS Engine Spatial extension 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.

If you are unfamiliar with the nmake utility, see the Microsoft Developer Network for further information.

The following files are provided for your use if you choose another compilation option: Makefile.Solaris and Makefile.Solaris.Template, Makefile.Linux and Makefile.Linux.Template, vs6.dsp and vs6.dsw, and vs7.sln and vs7.vcproj.

Creating your build environment

This scenario uses nmake to build and deploy its application. 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.

Replacing all instances of basic_sample *in your Makefile.Windows file with* RasterSlope *will complete steps 3 and 5 through 7 listed at right.*

7. Update the dependencies line from basic_sample.obj to be for RasterSlope.obj and to depend on RasterSlope.cpp and RasterSlope.h.

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.
```

To compile with another development environment, see Chapter 4, 'Developer environments', or the online Help.

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)
#
```

Equivalent lines will be found in the makefiles for each supported platform.

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
```

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 that runs the Visual Studio 6.0 version of vcvars32.bat and opens a command prompt that 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'.

This will have the same effect as typing "RasterSlope.exe C:\MyComputer\Rasters\RasterDataset tempslope" with command-line arguments at the command line.

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 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 standard C++ 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 was 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 pathname and filename of the input in separate locations. 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 filename.

```
#ifndef __PATHUTILITIES_ESRISCENARIO_H__
#define __PATHUTILITIES_ESRISCENARIO_H__
```

```
#include <iostream>
#include <ArcSDK.h>

// Extract the shapefile name from the full path of the file.
HRESULT GetFileFromFullPath(const char* inFullPath, BSTR* outFileName);

// Remove the filename 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 filename 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 filename. 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 on 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

To use ArcGIS Engine, it must be initialized and the proper files included. When done using ArcGIS Engine, it must be uninitialized.

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.

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>
```

2. *AoExit()* must be called before the application is exited. This allows portability to supported operating systems that 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 shut down the engine. These are general functions that 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

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 ArcGIS Engine Runtime license, it must use the 3D or Spatial extensions for ArcGIS Engine if required. If the application initially accesses an ArcGIS Desktop license (ArcView, ArcEditor, or ArcInfo), it must use ArcGIS 3D Analyst or ArcGIS Spatial Analyst extension licenses if required.

license being used, Engine or Desktop, either the Spatial extension for ArcGIS Engine Runtime or an ArcGIS Spatial Analyst extension must also be available. Your application must confirm the availability of, 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, shut down 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. No suitable license found."
                  << 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, it is now performing the license checking.

Computing the slope

At this point, you've determined the dataset on which to compute the slope and accessed ArcGIS Engine. Now, the slope calculation 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, 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 end user's machine. Doing so requires the following:

- An installation of ArcGIS Engine Runtime with Spatial extension on the user machine.
- The user's machine will need to have its ArcGIS Engine Runtime initialized.
- A copy of the application's executable, created at compile time, residing on the end user's machine.

Once these requirements are in place, your end user will be able to create a slope file for any dataset just by typing the following at the command line:

The output slope file is created in the same directory as its parent raster file.

```
RasterSlope <inputRaster> <outputRaster>
```

where `inputRaster` is the full path (including filename) to the raster datafile and `outputRaster` is the name of the output slope file to be created.

Your end user will have the option of compiling with any supported compiler, as you did during this scenario.

Alternatively, your end user 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.
- Burrough, Peter A., and Rachel A. McDonnell. *Principles of Geographical Information Systems*. Oxford University Press. 1998.

A

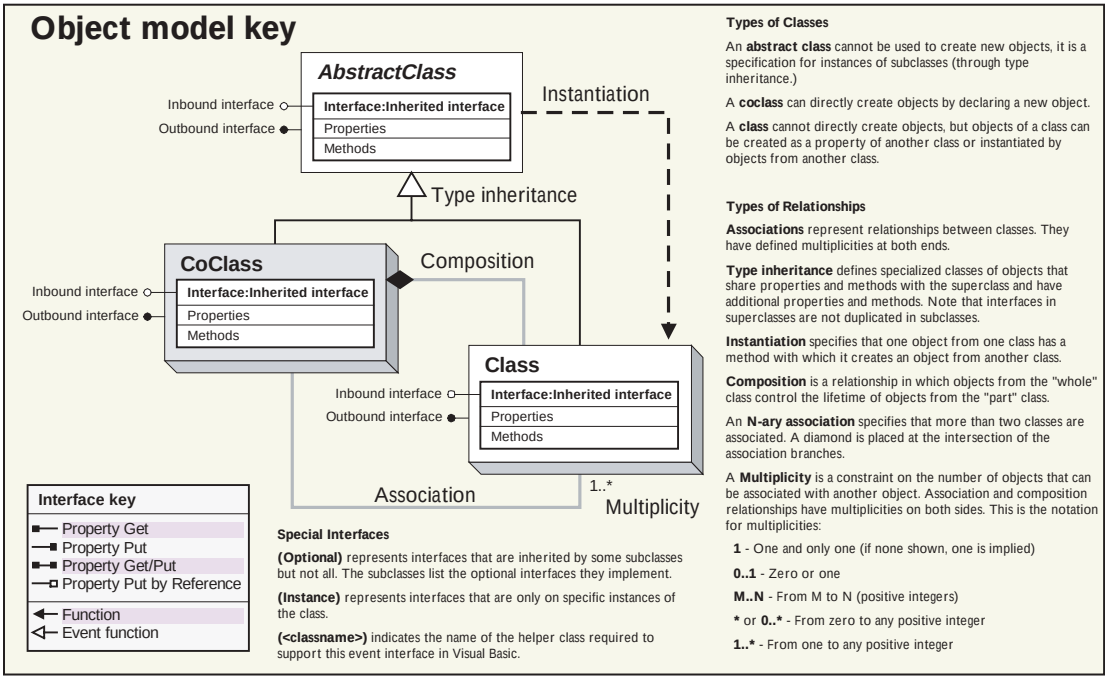
Reading the object model diagrams

The ArcObjects object model diagrams (OMDs) are an important supplement to the information you receive in object browsers.This appendix describes the diagram notation used throughout this book and in the object model diagrams that are accessed through ArcGIS Developer Help.

You can access the object model diagrams from the ArcGIS Developer Help system's contents pane by clicking Library Reference, the library of choice, and the Object Model Diagram link; or from their file location, by default <install_location>\DeveloperKit\Diagrams.

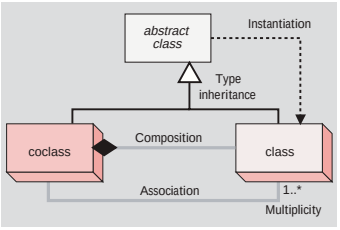
The diagram notation used in this book and the ArcObjects object model diagrams is based on the Unified Modeling Language (UML) notation, an industry diagramming standard for object-oriented analysis and design, with some modifications for documenting COM-specific constructs.

The object model diagrams are an important supplement to the information you receive in object browsers. Your development environment, Visual Basic or other, lists all of the classes and members but doesn't show the structure or relationships of those classes. These diagrams complete your understanding of the ArcObjects components.



Object model diagram key showing the types of ArcObjects and the relationships between them.

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.

```
Dim pWSName as iworkspacename        'Declare the interface you will use to
                                      access the object
Set pwsname = new workspacename      'Creates a new instance of the
                                      WorkSpaceName class

Dim pPoint as iPoint
Set pPoint  = new Point


Dim pGeometry as iGeometry
Set pGeometry = new Point
```

A *class* can't directly create new objects, but objects of a class can be created as a property of another class or by functions from another class.

```
Dim pName as IName
Dim pFeatClass As IFeatureClass
Set pFeatClass = pName.Open
```

The example discussed here—primary and secondary line classes that each meet the specification of the abstract line class—is an illustration of type inheritance. Type inheritance is discussed in detail later in this appendix.

An *abstract class* can't 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 want to create a subclass of their own; they show which interfaces are required and which are optional for the type of class 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.

```
Dim pGeometry as IGeometry
Set pGeometry = New Point
If typeof pGeometry is IGeometry the
  Msgbox "This is a Geometry object"
End if
```

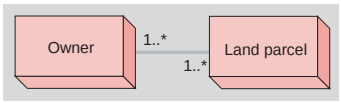
RELATIONSHIPS

To keep the object model diagrams as simple and usable as possible, only key relationships, or associations, are shown.

Among abstract classes, coclasses, and classes, there are several types of class relationships possible—associations, type inheritance, instantiation, composition, and n-ary associations.

Associations

Associations represent relationships between classes. They have defined multiplicities at both ends.



In this diagram, an owner can own one or many land parcels, and a land parcel can be owned by one or many owners.

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.

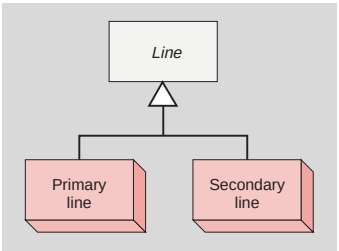
M..N—From M to N (positive integers).

* or 0..*—From zero to any positive integer.

1..*—From one to any positive integer.

Type inheritance

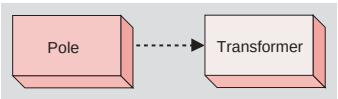
Type inheritance defines specialized classes that share properties and methods with the superclass and have additional properties and methods.



This diagram shows that a primary line (creatable class) and secondary line (creatable class) are types of a line (abstract class).

Instantiation

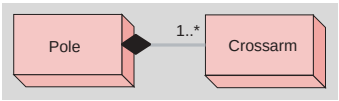
Instantiation specifies that one object from one class has a method with which it creates an object from another class.



A pole object might have a method to create a transformer object.

Composition

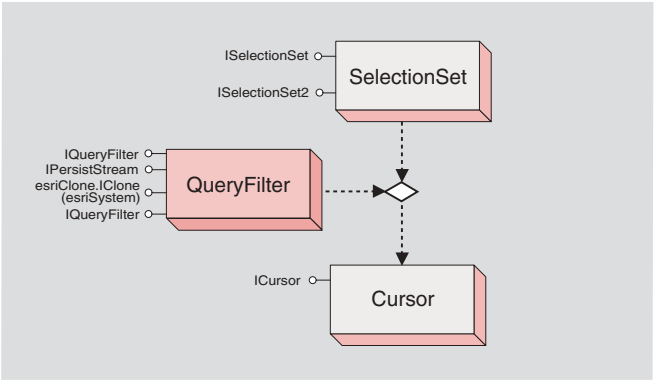
Composition is a stronger form of aggregation in which objects from the "whole" class control the lifetime of objects from the "part" class.



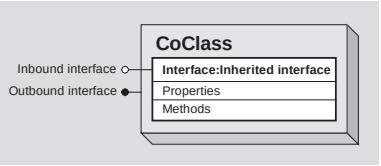
A pole contains one or many crossarms. In this design, a crossarm can't be recycled when the pole is removed. The pole object controls the lifetime of the crossarm object.

N-ary association

An *n-ary association* specifies that more than two classes are associated. A diamond is placed at the intersection of the association branches.



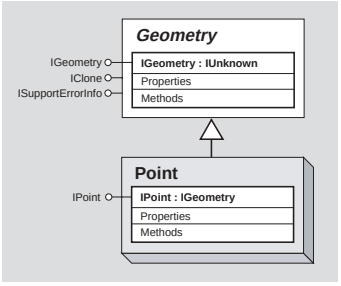
Interfaces are the access points for development with COM objects. There are inbound interfaces, which expose the properties and methods of a class, and outbound interfaces, which allow the class to interact with other classes.



A detailed discussion of COM and the IUnknown *interface can be found in the Microsoft Component Object Model section of Chapter 4,'Developer environments'.*

INTERFACE INHERITANCE

Since interfaces in ArcObjects are COM interfaces, they all inherit from *IUnknown*, the basis for COM itself. Additionally, as illustrated in the diagrams, some interfaces inherit from other ArcObjects interfaces. If one interface is inherited by another interface, then the members of the initial interface are also members of the inheriting interface. For example, since *IPoint* inherits from *IGeometry*, the members of the *IGeometry* interface are also members of the *IPoint* interface. This inheritance allows you to access the *IPoint* interface and use the members of *IGeometry* directly, without needing to query interface to the *IGeometry* interface.



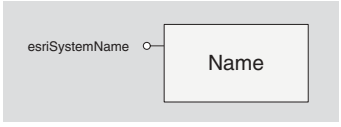
IGeometry *inherits from* IUknown *and, in turn,* IPoint *inherits from* IGeometry

Interface inheritance is used extensively to, in effect, add functionality to existing interfaces. Although the rules of COM dictate that an interface signature, once deployed, can't change, a new interface can be created that inherits from the original interface. For example, the *IEditor2* interface extends the *IEditor* interface with additional members.

INBOUND INTERFACES

Some inbound interfaces are shown on the diagrams with special notations that provide information in addition to the usual list of members.

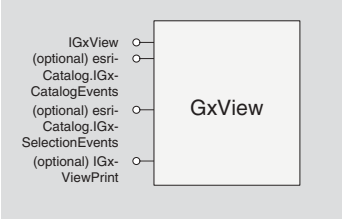
Interfaces defined in other libraries



The Name *abstract class and the interface it implements*

If the interface name is prefixed with a library name, such as *esriSystem.IName*, then the interface is inherited from a library other than the one implementing it. The library name reflects the library in which the interface is defined. In the column at left, the *Name* abstract class, an object in the GeoDatabase library, is shown. As shown by the library name prefix, the interface implemented by *Name* is actually defined in the System library.

Optional interfaces



The GxView *class includes a number of interfaces that implemented optionally.*

Some interfaces can be inherited optionally by other classes. For example, abstract classes can have optional interfaces that may be included or excluded from its subclasses. These are designated by the prefix (Optional). If you are creating your own *GxView* class, you don't need to implement the *IGxViewPrint* interface to be a *GxView* class; however, you do need to implement the *IGxView* interface.

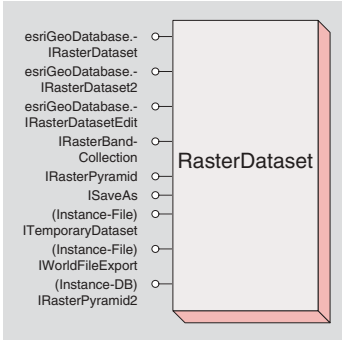
As a developer, if you intend to utilize an optional interface, you must verify that the interface was implemented by the object with which you are working. Attempts to access an optional interface that is not implemented will produce the

error message “Run-time error ‘13’ ; Type mismatch”.

```
Dim pGxApp As IGxApplication
Set pGxApp = Application
Dim pGxV As IGxView
Set pGxV = pGxApp.TreeView
'Optional interface for GxViews
Dim pGxVP As IGxViewPrint
'Attempting to QI to this will result in a runtime error
Set pGxVP = pGxV
```

Interfaces implemented in select instances

Some classes have been designed to have varying implementations instead of having multiple classes that inherit from a single base or abstract class. In these cases, certain interfaces are implemented in select instances. For example, the *RasterDataset* class can be instantiated by different *Workspace* classes depending on the type of data being accessed. When file-based data is used to instantiate a *RasterDataset* class, the *ITemporaryDataset* interface is implemented; however, if ArcSDE software-based data is used to instantiate the *RasterDataset* class, the *IRasterPyramid2* interface isn’t implemented.



The RasterDataset *class and the interfaces it implements when instantiated by file-based and ArcSDE software-based data*

Interfaces that are implemented in select instances are designated with the prefix (Instance). Attempts to access a selected instance interface that hasn’t been implemented will produce a “Run-time error ‘13’ ; Type mismatch”.

```
Dim pWsFact As IWorkspaceFactory
Dim pWs As IRasterWorkspace
Dim pRasterDataset As IRasterDataset
'Open the workspace
Set pWsFact = New RasterWorkspaceFactory
Set pWs = pWsFact.OpenFromFile("D:\data\canada", 0)
'Open the raster dataset
Set pRasterDataset = pWs.OpenRasterDataset("Dem")
'Test if interface is implemented before QI
Dim pTempDS As ITemporaryDataset
If TypeOf pRasterDataset Is ITemporaryDataset Then
  Set pTempDS = pRasterDataset
End If
```

OUTBOUND INTERFACES

Outbound interfaces, also known as event interfaces, provide notification when a certain event occurs. Interaction with the members of an outbound interface requires that another object exists to catch the event occurrences; this is commonly referred to as an event sink. Event sinks have code that responds when certain events occur, an editing operation, for example. In Visual Basic, the creation of classes that will act as sinks to an outbound interface requires that the outbound interface declaration include the name of the class that implements the interface.

```
Private WithEvents EditEvents as Editor
```

Although the object variable has been declared with the class that will be raising the events, it points to nothing. The object variable must be initialized using the

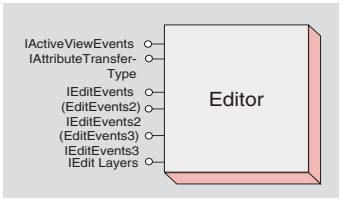
Set statement linking it to the existing *Editor* object.

```
Public Sub SetEvents()
  Dim pID as New UID
  pID = "esriEditor.Editor"
  Set m_pEditor = Application.FindExtensionByCLSID(pID)
  Set EditEvents = m_pEditor
```

Secondary outbound interfaces (nondefault)

When a class implements more than one outbound event, the secondary interface name is preceded by a classname, *(EditEvents2)IEditEvents2* for example, where the classname indicates the name of a Helper class. The Helper class is an artificial coclass that solves the Visual Basic problem with multiple outbound interfaces on an object. When working with a nondefault outbound interface for *EditEvents2* in Visual Basic 6, declare the variable as follows:

```
Private WithEvents pEditEvents2 as EditEvents2
```

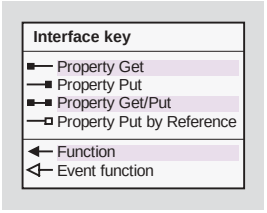


Editor *class contains several outbound interfaces*

INTERFACE MEMBERS

The members of an interface include its properties, which specify the state of an object, and its methods, which perform some action.

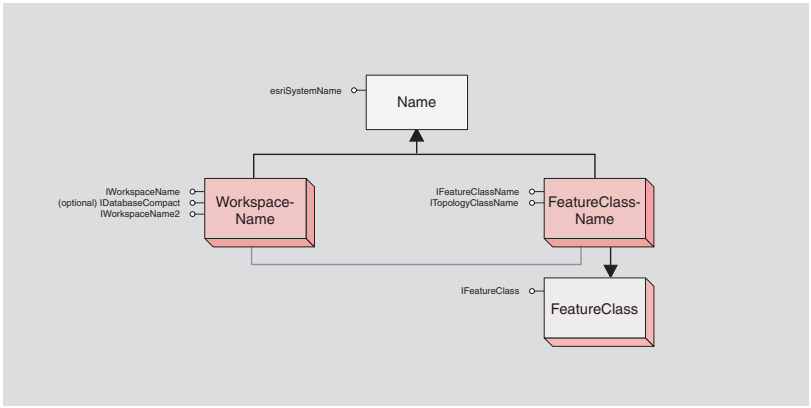
Properties are designated as read-only (*get*), write-only (*put*), or read–write (*get/put*). Additionally, the value of a property may be a simple data type, a long for the x-value of a point object, or another class, such as a coordinate system class (*GeographicCoordinateSystem*, for example) for the *SpatialReference* property.



Properties and their symbols

Each property on the diagram is listed with the data type required or returned. The symbolization and syntax of properties that can be set with an object will vary based on whether the property is put by value (*put*) or put by reference. If ObjectG is assigned to a property by value on ObjectM, then ObjectG is contained within ObjectM. When an object is passed by reference, an association is formed between the two objects. This is advantageous because an object can be reused in many associations, using less memory space.

```
Dim psphere As ISphere
Set psphere = New Sphere

Dim ppoint As IPoint
Set ppoint = New point
psphere.Center = ppoint

Dim pspatref As ISpatialReference
Set pspatref = New GeographicCoordinateSystem

'This won't compile
'psphere.SpatialReference = pspatref
Set psphere.SpatialReference = pspatref
```



In this example, the abstract class *Name* implements the *IName* interface; its inheritance relationship to all other name objects indicates that they each implement the *IName* interface as well. Although neither the *FeatureClassName* nor *WorkspaceName* coclass shows the *IName* interface in the diagrams, you know it is there because of this inheritance. The *Open*() method on the *IName* interface is used to instantiate the *FeatureClass* object. This usage guarantees that the new *FeatureClass* object is properly created and includes all necessary information.

The case below illustrates its importance clearly: opening a shapefile dataset to extract a feature class is not as simple as just reading the database records.

```
Dim pwrkspc As IWorkspaceName
Set pwrkspc = New WorkspaceName
pwrkspc.PathName = "D:\data\canada"
pwrkspc.WorkspaceFactoryProgID = _
"esriDataSourcesFile.shapefileworkspacefactory.1"
'*****************************************************
Dim pdatasetname As IDatasetName
Set pdatasetname = New FeatureClassName
pdatasetname.Name = "Canada.dbf"
Set pdatasetname.WorkspaceName = pwrkspc
Dim pname As IName
Set pname = pdatasetname
Dim pfeatclass As IFeatureClass
Set pfeatclass = pname.Open
''Check FeatureType property to ensure you have a featureclass object
MsgBox pfeatclass.FeatureType
```

An association is shown in the code above where the *WorkspaceName* object was set to the *WorkspaceName* property of the *FeatureClassName* object. As noted earlier, for simplicity's sake, many of the associations in ArcObjects aren't drawn on the diagrams. In fact, this association between the *WorkspaceName* and *FeatureClassName* classes isn't shown on the GeoDatabase library object model diagram; however, it can be seen in the *IName* interface detail for the *WorkspaceName* property since the symbol used is a Property Put by Reference. Since the *WorkspaceName* class is being held as a reference and not in a composition relationship, the object's lifespan will not be controlled by the *FeatureClassName* object. If you were to use the code below to set the *FeatureClassName* object to nothing, the *WorkspaceName* class would still exist.

```
Set pfeatclass = Nothing
If Not pwrkspc Is Nothing Then
   MsgBox "Object still exists"
End If
```

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*).*

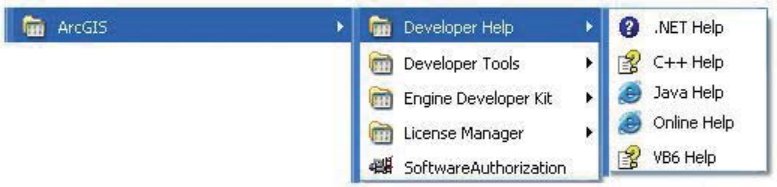
A typical SDK installation

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.

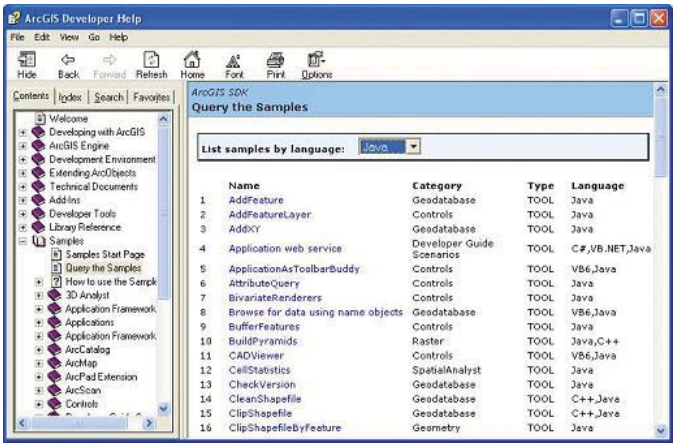


The help systems reside in the DeveloperKit\Help folder but are typically launched from the start menu or F1 help in Visual Basic 6 and Visual Studio .NET 2003. The graphic to the left shows the start menu options for opening the help systems.

SAMPLES

The ArcGIS developer kit contains more than 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 Samples under Developer Kit.

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.

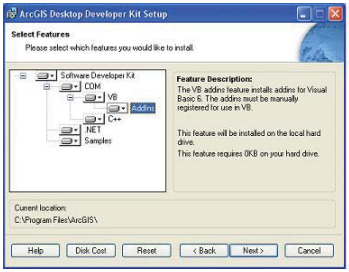
Additional tools available in the Desktop developer kit

- 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



Installation dialog box for selecting the Visual Basic add-ins.

The following Visual Basic 6 add-ins are available but only installed if you select them during the installation process:

- 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.
- ESRI License Initializer—Automatically generates and adds license initialization code to an ArcObjects project.

Visual Studio .NET add-ins

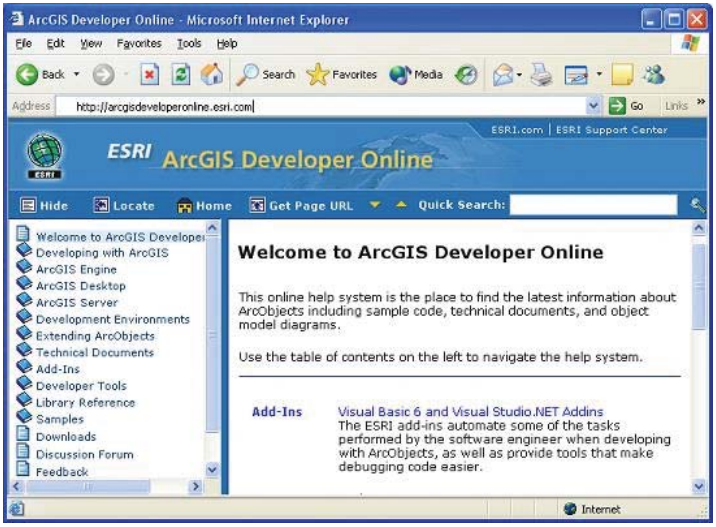
The following .NET add-ins are automatically installed during setup if a version of Visual Studio .NET 2003 is detected:

- ESRI Component Category Registrar—Stubs out registration functions to enable self component category registration.
- ESRI Guid Generator—Inserts a GUID attribute.
- ESRI Automatic References—Automatically adds ArcGIS library references.
- ESRI License Initializer—Automatically generates and adds license initialization code to an ArcObjects project.

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*).



C

Converting personal geodatabases

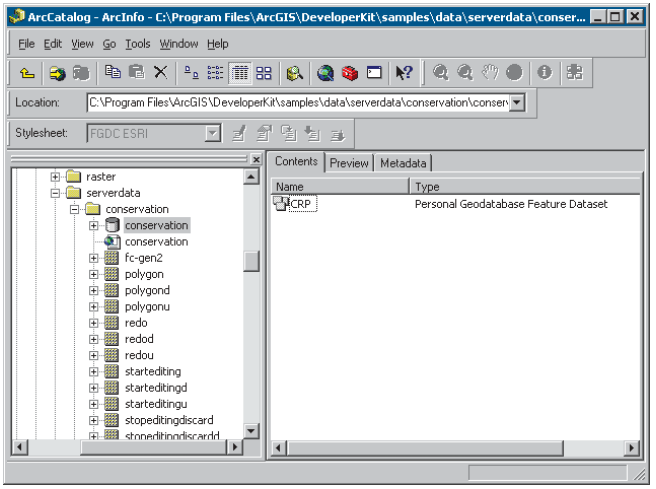
There are two kinds of geodatabases: personal geodatabases and multiuser geodatabases. Multiuser geodatabases are also known as ArcSDE-based geodatabases. While ArcGIS Engine is supported on UNIX, these platforms do not support the use of personal geodatabases (.mdb). Your personal geodatabases must be converted to an ArcSDE geodatabase or a supported file-based format, such as shapefiles, coverages, or raster data, before you can use that data to serve a map on Solaris or Linux.*

ArcCatalog allows you to copy data from a personal geodatabase and paste it to an ArcSDE geodatabase. You can copy entire feature datasets or individual feature classes, raster catalogs, or individual rasters and tables. For every feature dataset, feature class, raster catalog, or individual raster or table you copy and paste, the result is an equivalent feature class, raster catalog, and individual raster or table in the destination geodatabase with all the features or records from the source data.

This appendix describes the steps involved in this process as well as the procedure to convert a personal geodatabase to a file-based shapefile, coverage, or individual raster.

COPYING A PERSONAL GEODATABASE AS AN ARCSDE GEODATABASE

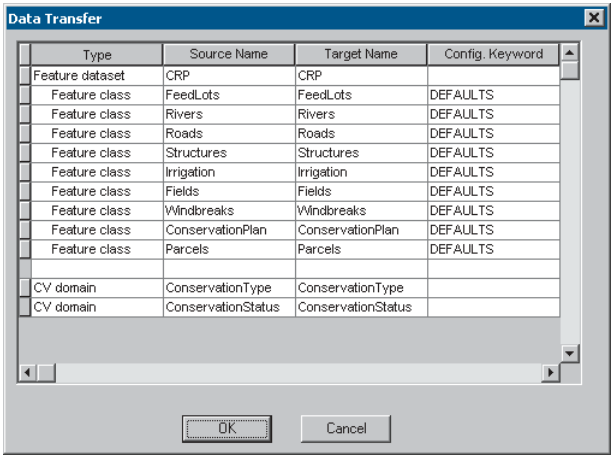
1. Open ArcCatalog.
2. In the tree view or contents view, browse to the location where the personal geodatabase (*.mdb) is stored. If the file is located on a UNIX machine, you should have access to that file from Windows and have sufficient permissions for reading the data.
3. Select the personal geodatabase to be copied. Right-click the feature dataset, feature class, raster data, and table you want to copy.
4. Click Copy.



You can also copy and paste data by clicking the data in the ArcCatalog tree or contents view, dragging it to another location, and dropping it.

To copy a geometric network or a topology class and all the participating feature classes, copy and paste the network or topology class only. This will copy all the participating feature classes along with it. You cannot copy and paste individual feature classes participating in a network or topology.

5. Right-click the ArcSDE geodatabase you want to copy the data to.
6. Click Paste. A dialog box appears that indicates what data is being copied. Any name conflicts are automatically resolved and highlighted in red.

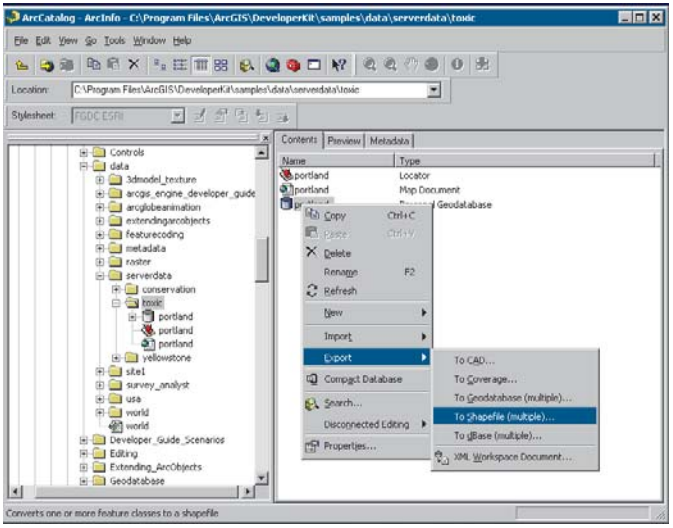


7. Type over the target name to change any of the resolved names.
8. If you want to control how the new feature classes and tables are created and stored, click a keyword and choose a new one from the dropdown list.
9. Click OK to copy the data into new feature classes and tables.

EXPORTING FROM A PERSONAL GEODATABASE FEATURE DATASET OR FEATURE CLASS TO A SHAPEFILE OR COVERAGE

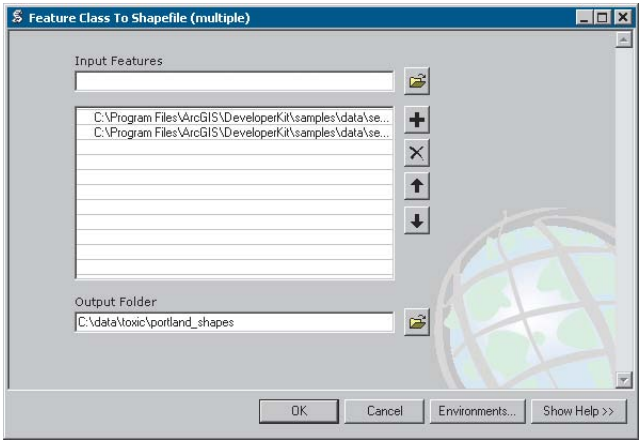
1. Open ArcCatalog. (This step is not required if the data is converted to an ArcSDE Feature class.)
2. In the Catalog tree or contents view, browse to the location where the personal geodatabase (*.mdb) is stored. If the file is located on a UNIX machine, you should have access to that file from Windows and have sufficient permissions for reading the data.
3. Right-click the personal geodatabase to be exported. Click Export and To Shapefile (Multiple) or To Coverage depending on your preference.

   This will export all feature classes contained in the personal geodatabase. To export tables, select the tables to be exported and click the option To dBase (multiple).



4. A dialog box appears that indicates what data is being copied. Enter the location of output directory where the shapefiles or coverages will be created.

   As this data is to be used with ArcGIS Engine on UNIX/Linux, be aware that the name of the data and path to the data should be all lowercase.



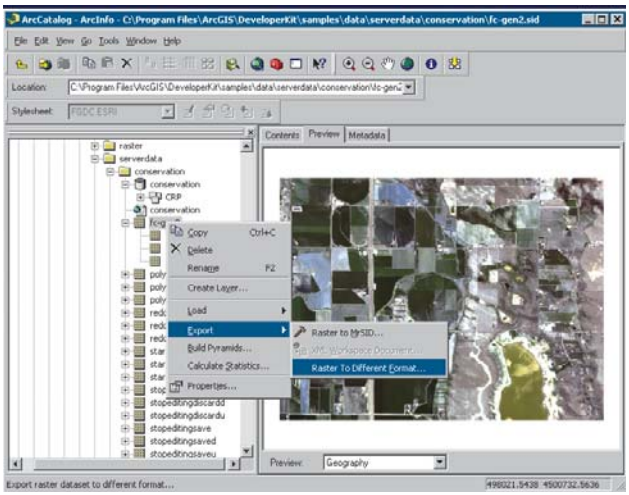
5. Click OK to copy the data into shapefile or coverage features and .dbf tables.

EXPORTING FROM PERSONAL GEODATABASE RASTER DATA TO A FILE-BASED RASTER FORMAT

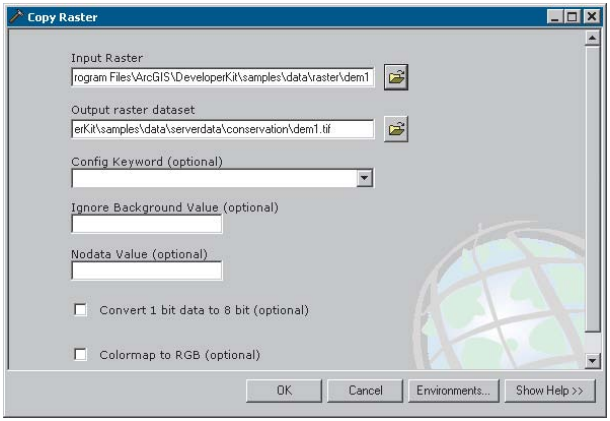
ArcCatalog can be used to convert personal geodatabase raster data into a TIFF, ERDAS IMAGINE, ESRI Grid, or MrSID raster format.

1. Open ArcCatalog. (This step is not required if the data is converted to ArcSDE raster.)
2. In the Catalog tree or contents view, browse to the location where the personal geodatabase (*.mdb) is stored. If the file is located on a UNIX machine, you should have access to that file from Windows and have sufficient permissions for reading the data.

3. Right-click the personal geodatabase raster to be exported. Click Export and click Raster To Different Format for TIFF, ERDAS IMAGINE, and ESRI Grid formats (choose Export and Raster to MrSID for exporting to MrSID imagery).



4. A dialog box appears that indicates the input raster and an option to choose the output directory where the output raster will be created. Choose the output folder name, filename for raster data, and extension. Depending on the extension of the filename (*.img, *.tif), an ERDAS IMAGINE or TIFF image will be created. When the extension is left blank, an ESRI grid file will be created.



Updating links to data in an ArcMap document

1. Open the map document in ArcMap.
2. For the layer that requires update in data path, open the layer properties dialog box (right-click the layer and click Properties). Use the Set Data Source button on the Source tab of the layer's Properties dialog box and choose the desired ArcSDE feature, shapefile, or coverage data. If using shapefile or coverage data, make sure the data path and filenames are all in lowercase.
3. Repeat step 2 for each layer.
4. Save the map document.

   If using relative paths to data, then the map is ready to serve with ArcGIS Engine. Fully qualified UNIX paths to the data are also supported. For additional information on updating Windows paths to UNIX, refer to the topic 'Data and cross-platform development' in ArcGIS Developer Help.

For information on related topics, see 'How to convert geodatabase annotation to coverage annotation', 'Copying feature datasets, classes, and tables to another geodatabase', and 'Exporting raster formats' in the ArcGIS Desktop Help.

D

Installing ArcGIS Engine Runtime on Windows, Solaris, and Linux

In most cases you will develop applications that, at minimum, initialize against the ArcGIS Engine Runtime product and extension licenses. Under such circumstances, the next step in the deployment process for your application is the installation of ArcGIS Engine Runtime. The ArcGIS Engine Runtime setup program is available for installation on Microsoft Windows, Sun Solaris, and Linux operating systems. This appendix briefly introduces the technology used to create the installation program for ArcGIS Engine Runtime on the applicable operating system and discusses the various installation mechanisms for the runtime on either Solaris and Linux or Windows platforms.

This section specifically discusses installation on Windows; the next discusses installation on Solaris and Linux.

ABOUT THE ARCGIS ENGINE RUNTIME INSTALLATION PROGRAM

More information on Windows Installer can be found in the Windows Installer Software Development Kit. Download the kit at http://www.microsoft.com/msdownload/platformSDK/sdkupdate/.

The ArcGIS Engine Runtime installation program, or setup, was created using Windows Installer (MSI) technology from Microsoft. This technology uses a package file (.msi) and a client-side installer service (msiexec.exe). 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.

An MSI-based setup consists of features. A feature is an individual portion of the application's total functionality that may be installed independently. The ArcGIS Engine Runtime setup consists of the following installation features:

The available ArcGIS Engine Runtime setup features are illustrated in the table on the right.

Feature	Descriptive Feature Name	Description
ArcEngine	ArcGIS Engine	ArcGIS Engine
JavaRuntime	ArcGIS Engine—Java Runtime	Java Archives
DotNetRuntime	ArcGIS Engine .NET Runtime	.NET Assemblies

All ArcGIS Engine-based applications depend on an installation of the ArcEngine feature. Applications built using the Java and .NET APIs for ArcGIS Engine require installations of the JavaRuntime and DotNetRuntime features, respectively, in addition to the ArcEngine feature.

Solaris and Linux are also supported. For details on the system requirements for these operating systems, see the next section, 'Installing ArcGIS Engine Runtime on Solaris and Linux'.

ArcGIS Engine Runtime is supported on 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*.

INSTALLING ARCGIS ENGINE RUNTIME

You cannot redistribute individual ArcGIS Engine Runtime files; the installation mechanisms discussed in this appendix are the only means of installing ArcGIS Engine Runtime files.

As mentioned above, ArcGIS Engine software-based applications require that ArcGIS Engine Runtime be installed on the end user's machine. Installation of the runtime can be handled in either of two ways:

1. Your end user runs the ArcGIS Engine Runtime setup directly from the CD.
2. You include the ArcGIS Engine Runtime setup within your own application's installation program.

The following sections document the general requirements for the installation of the ArcGIS Engine Runtime, no matter which of the two types of installation mechanism you choose, and the steps necessary to install the runtime successfully using your chosen mechanism.

General requirements

Irrespective of the installation method you chose for ArcGIS Engine Runtime, you should be aware of the following:

You can obtain a redistributable version of the .NET Framework 1.1 from the Microsoft Web site at http://msdn.microsoft.com/netframework/downloads/framework1_1redist/.

- **Prerequisites—**If your ArcGIS Engine-based application was built using the .NET API, the .NET Framework 1.1 must be installed on your user's machine prior to installing the ArcGIS Engine Runtime.
- **Older versions of ArcGIS software—**ArcGIS Engine Runtime cannot be installed on any machine that already contains versions of ArcGIS products prior to the version you will be installing. This includes ArcIMS, ArcGIS Desktop, ArcGIS Workstation, ArcReader, ArcGIS Engine Runtime, ArcGIS Server, .NET ADF, and Java ADF.
- **Existing installation of ArcGIS Engine Runtime—**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.
- **Installation location—**Each of the ArcGIS 9 products (ArcGIS Engine Runtime, ArcGIS Engine Developer Kit, ArcGIS Desktop, ArcReader standalone, and ArcGIS Server) installs to the same installation directory. The first ArcGIS 9 product installed determines the installation location for all subsequent ArcGIS 9 products. For example, if ArcGIS Desktop is installed to C:\Desktop, the installation location for all ArcGIS 9 products is C:\Desktop\ArcGIS. If ArcGIS Engine Runtime is installed after ArcGIS Desktop, the opportunity to browse to an installation location is not provided. The initial ArcGIS Desktop installation has predetermined the installation location for all future ArcGIS 9 products. Therefore, in this example, ArcGIS Engine Runtime will also be installed to C:\Desktop\ArcGIS.
- **Disk space—**If you run out of disk space while installing an ArcGIS 9 product, you must uninstall all ArcGIS 9 products and reinstall them to a location where more disk space is available. ArcGIS 9 products (excluding ArcSDE, ArcIMS, and ArcInfo Workstation) cannot be installed to different locations.

End user installs ArcGIS Engine Runtime on their system

For additional ArcGIS Engine Runtime CDs, contact ESRI Customer Service at www.esri.com, *or in the USA call 888-377-4575, or contact your local ESRI regional office.*

In this case, your user installs the ArcGIS Engine Runtime either directly from the CD ESRI provided with the ArcGIS Engine product or from a CD that you created by copying the contents of the ArcGIS Engine Runtime CD image to it.

You should distribute the following steps or similar instructions to your users:

You cannot redistribute individual ArcGIS Engine Runtime files; the installation mechanisms discussed in this appendix are the only means of installing ArcGIS Engine Runtime files.

1. Confirm that any currently installed ArcGIS products are the same version as the Engine Runtime setup to be installed. To check if any of these products are installed on the machine and verify the version number:

   a. Click Start, Control Panel, and open the Add/Remove Programs dialog box.

   b. If ArcGIS Engine Runtime is listed in the programs list, it has been installed on the machine. Confirm that ArcGIS Engine Runtime is the correct version. Click Support Information to determine the product version number. Close the window and proceed to the installation steps for <your application name here> if needed.

c. If no ArcGIS products are shown in the programs list, proceed to Step 2. If ArcGIS products are shown in the programs list, click Support Information to view the version of the product.

d. If the versions displayed are the same, proceed to Step 2. If the versions displayed are not the same, the products need to be removed before installing ArcGIS Engine Runtime. Do not proceed to Step 2 until these products have been uninstalled.

2. Launch the ArcGIS Engine Runtime installation program by navigating to the CD drive location and double-clicking the setup.exe file.

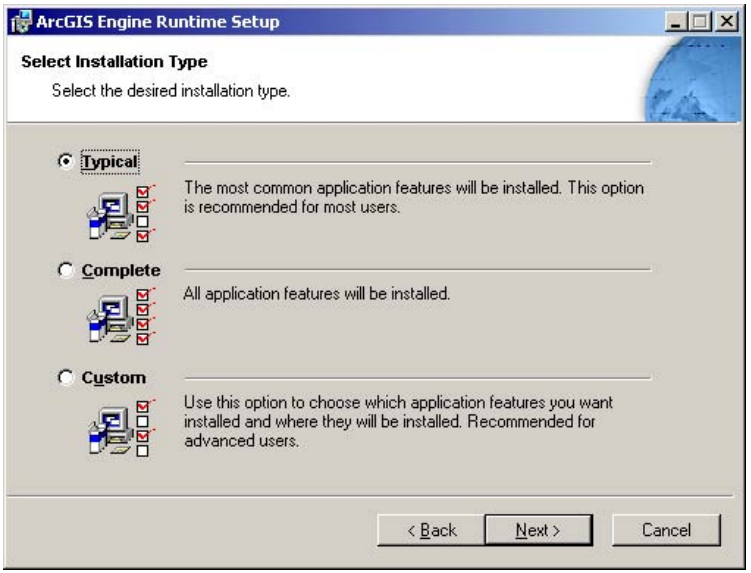
If the ArcGIS Engine Runtime setup is launched, and ArcGIS Engine Runtime already exists on the machine, the setup will execute as a maintenance installation.

3. In the installation, click Next if all Windows applications are closed.

4. Review and accept the license agreement and click Next to proceed with the installation.

5. Choose the installation type and click Next to continue.

The Select Installation Type panel of the ArcGIS Engine Runtime installation wizard is illustrated on the right. The possible types are discussed below.



A Typical installation includes the ArcEngine feature. In addition, if the .NET Framework is detected, the DotNetRuntime feature will be installed.

When Complete is selected all available ArcGIS Engine Runtime features are installed. If the .NET Framework prerequisite is not detected, the DotNetRuntime feature will be excluded.

If your ArcGIS Engine-based application utilizes Java, your user can select Custom and manually choose the JavaRuntime feature. As noted above a Complete installation also installs the JavaRuntime feature.

If the ArcGIS Engine-based application to be installed requires Java, choose Custom to manually select the JavaRuntime feature or choose Complete to install all ArcGIS Engine Runtime installation features.

6. Follow any further instructions of the ArcGIS Engine Runtime installation until it indicates that the installation is complete. If you installed, and intend to use, the Java feature, proceed to Step 7. If not, the installation of ArcGIS Engine Runtime is now complete. Proceed to the installation steps for <your application name here> if needed.

If the JavaRuntime feature was not installed during the initial installation of ArcGIS Engine Runtime, it can be installed at a later date by opening the Add/Remove Programs dialog box, clicking Change on the ArcGIS Engine Runtime listing, then clicking Modify. When the Select Features dialog box opens, right-click on the Java feature and choose Will be installed on local hard drive.

7. The JavaRuntime feature of ArcGIS Engine Runtime requires some additional postinstallation steps to be performed.
   a. Click Start, Control Panel, and System. Double-click System Properties to open its dialog box.
   b. Click Environment Variables on the Advanced tab.
   c. In the System variables window, click the Path variable and click Edit.
   d. Scroll to the end of the Variable value text box, and add the path to the ArcGIS\Bin directory. Separate it from previous entries with a semicolon.
   e. Continuing on at the end of the same text box, use a semicolon separator and add in the path to the ArcGIS\Java\jre\bin directory.

   The installation of ArcGIS Engine Runtime is now complete. Proceed to the installation steps for <your application name here> if needed.

Installing an ArcGIS Engine Runtime service pack

ArcGIS service packs are cumulative.

If your application also requires an ArcGIS Engine service pack, you can either instruct your user to download the service pack from *http:\\support.esri.com*, or you can provide the service pack (.msp file) on the ArcGIS Engine Runtime CD that you provided.

You should distribute the following steps or similar instructions to your users (the steps assume ArcGIS Engine Runtime is already on the target machine.):

1. Check whether the ArcGIS Engine Runtime service pack required has been installed to the target machine by doing the following:
   a. Click Start > Run. Type "Regedit" to open the Registry Editor.
   b. Check for the SPNumber registry key under the following registry hive: HKEY_LOCAL_MACHINE\SOFTWARE\ESRI\. The SPNumber value reflects the service pack number installed.
2. Install the service pack using the following command line:

```
msiexec.exe /p <location of msp file>\ArcGIS<Product>.msp REINSTALL=ALL
REINSTALLMODE=omus
```

ArcGIS Engine Runtime setup is included in your application's installation program

The second way to distribute the ArcGIS Engine Runtime is for you to incorporate its setup into your own ArcGIS Engine software-based application's installation program. This is possible because ArcGIS Engine Runtime can be installed without a graphical user interface by running the setup using Windows Installer command-line parameters. This process utilizes 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 for the table on the right: 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 if 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 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.

These commands can be used to perform various installation functions. The features available to install are specified with the ADDLOCAL parameter. As shown earlier, 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

Some example functions and the commands used to achieve them follow:

- A typical installation to a nondefault location without a user interface

  ```
  Msiexec.exe /i <setup location>\setup.msi /qn InstallDir=C:\Mysetup
  ```

- A typical installation to a nondefault installation location with a basic user interface (progress bar)

  ```
  Msiexec.exe /i <setup location>\setup.msi /qb InstallDir=C:\Mysetup
  ```

- A complete installation to the default installation location without a user interface

  ```
  Msiexec.exe /i <setup location>\setup.msi /qn ADDLOCAL=All
  ```

- A custom installation without a user interface consisting of the ArcGIS Engine and .NET installation features

  ```
  Msiexec.exe /i <setup location>\setup.msi /qn
  ADDLOCAL=ArcEngine,DotNetRuntime
  ```

These command-line parameters can be used to include the ArcGIS Engine Runtime setup in your application's setup in any of the following three methods—at the end of an MSI-based setup, within a batch file, or within a scripted setup. Each will be discussed in more detail later in this section.

If your application requires an ArcGIS Engine Runtime service pack, the following standard command lines can be used to apply a service pack (these examples are for installing ArcGIS Engine Runtime 9.0 service pack 1):

- To apply a service pack to an existing ArcGIS Engine Runtime installation on a local machine:

  msiexec.exe /p <msp file location>\ArcGISEngine90sp1.msp REINSTALL=ALL REINSTALLMODE=omus

- To apply a service pack to an admin installation of ArcGIS Engine Runtime:

  msiexec.exe /a <location of admin installation>\setup.msi /p <msp location>\ArcGISEngine90sp1.msp

In addition to the general requirements listed earlier that apply to any installation of ArcGIS Engine Runtime, the following are applicable to installs that are incorporated within other applications:

- **Prerequisites**—The ArcGIS Engine Runtime setup uses Windows Installer 2.0. To utilize command-line parameters to install ArcGIS Engine Runtime, Windows Installer 2.0 must be installed and running on the target machine. The setup.msi file automatically checks for Windows Installer 2.0; if it is not detected a message box is displayed. Even if you are including the ArcGIS Engine Runtime setup in a non-MSI-based setup, Windows Installer 2.0 must be installed on the machine. The ArcGIS Engine Runtime setup uses the Windows Installer technology.

The Windows Installer 2.0 setup is available from <ArcGIS Engine Runtime CD>\Support\MSI\instmsiw.exe.

- **Compatible setup programs**—If the ArcGIS Engine Runtime setup will be launched at the end of an MSI-based setup, you must create your MSI setup using Windows Installer 2.0 or higher to be compatible with the ArcGIS Engine Runtime setup.
- **Nested MSIs**—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.

Your setup process should follow these steps:

1. Before launching the ArcGIS Engine Runtime setup, you need to check whether ArcGIS Engine Runtime and any needed optional features—JavaRuntime and DotNetRuntime—have already been installed to the target machine. If your application also requires an ArcGIS Engine Runtime service pack, you need to check that the current service pack is on the user's machine.

   a. For ArcGIS Engine Runtime, check for the following registry key:

   `HKEY_LOCAL_MACHINE\Software\ESRI\ArcGIS Engine Runtime`

   If the key exists, then ArcGIS Engine Runtime has already been installed on the target machine and you may not need to launch the ArcGIS Engine Runtime setup. Continue to Step 1b if your application requires the Java feature or Step 1c if your application requires the .NET feature; otherwise, proceed with the installation of your application.

   If the key does not exist, then ArcGIS Engine Runtime has not been installed. Continue to Step 1b if your application requires the Java feature or Step 1c if your application requires the .NET feature; otherwise, proceed to Step 2.

Your application setup program should not launch the ArcGIS Engine Runtime setup if it is already installed on the machine.

- If ArcGIS Engine Runtime 9.0 is required, the RealVersion registry key under HKEY_LOCAL_MACHINE\Software\ESRI\ArcGIS Engine Runtime\ will have a value of 9.0.

- If ArcGIS Engine Runtime 9.1 is required, the RealVersion registry key under HKEY_LOCAL_MACHINE\Software\ESRI\ArcGIS Engine Runtime\ will have a value of 9.1.

b. For Java, check the following registry entry:

For 9.0:

HKEY_CLASSES_ROOT\Installer\Features\7A1A3A9178A2BC74EB114EA6B5DB1C1B

For 9.1:

HKEY_CLASSES_ROOT\Installer\Features\BE4BB2A7FG2185944AB7F1350406D01A

The JavaRuntime string value represents the Java installation feature of ArcGIS Engine Runtime.

If the value displayed is

"JavaRuntime"="ArcEngine"

then JavaRuntime *is* installed. Proceed to Step 1c if your application requires the .NET feature; otherwise, proceed with the installation of your application.

If the value displayed is

"JavaRuntime"="|ArcEngine"

or

"JavaRuntime"="□ArcEngine"

then JavaRuntime *is not* installed. Proceed to Step 1c if your application requires the .NET feature; otherwise, proceed to Step 2.

c. For .NET, check the following registry entry:

For 9.0:

HKEY_CLASSES_ROOT\Installer\Features\7A1A3A9178A2BC74EB114EA6B5DB1C1B

For 9.1:

HKEY_CLASSES_ROOT\Installer\Features\BE4BB2A7FG2185944AB7F1350406D01A

The DotNetRuntime string value represents the .NET installation feature of ArcGIS Engine Runtime.

If a DotNetRuntime string value is not displayed under this registry key, then the prerequisite—.NET Framework 1.1—was not installed on the machine at the time the ArcGIS Engine Runtime setup was originally run.

If the value displayed is

"DotNetRuntime"="ArcEngine"

then DotNetRuntime *is* installed. Proceed with the installation of your application.

If the value displayed is

"DotNetRuntime"="|ArcEngine"

or

"DotNetRuntime"="□ArcEngine"

then DotNetRuntime *is not* installed. Proceed to Step 2.

Later sections provide examples illustrating each of these methods of installing the ArcGIS Engine Runtime.

The later section,'Launching the ArcGIS Engine Runtime installation within a batch file,' illustrates the usage of these commands.

If .NET Framework 1.1 is not detected on the machine, the DotNetRuntime feature will not be installed.

2. Launch the ArcGIS Engine Runtime setup at the end of an MSI-based setup, within a batch file, or within a scripted setup.

   If your application requires the Java Runtime or DotNetRuntime features of the ArcGIS Engine Runtime, you will need to install them 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 core ArcGIS Engine Runtime (ArcEngine feature) does not exist on the target machine, the setup will automatically install the ArcEngine feature as well as the selected optional feature. Java is an optional installation feature and is not installed with a typical ArcGIS Engine Runtime installation. The .NET feature, on the other hand, is included in the typical install if the .NET Framework is detected on the target machine.

Launching the ArcGIS Engine Runtime installation at the end of an MSI-based setup

Launching the ArcGIS Engine Runtime installation program at the end of an MSI-based setup is an example of installing the necessary ArcGIS Engine runtime features without having your user perform the installation themselves directly from the ESRI-provided ArcGIS Engine Runtime CD. The following example illustrates this particular installation mechanism.

In example 1, the ArcGIS Engine Runtime setup launches once your end user clicks Finish to complete the installation of your application. The MSI authoring tools included with Wise for Windows Installer are used to add 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.

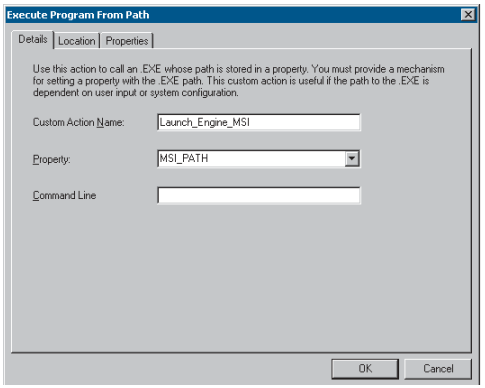
Example 1 (ArcGIS Engine Runtime 9.0 or 9.1 is required.)

Follow these steps to launch the setup.exe file for ArcGIS Engine Runtime when it's located in an ArcEngine folder on your application's media:

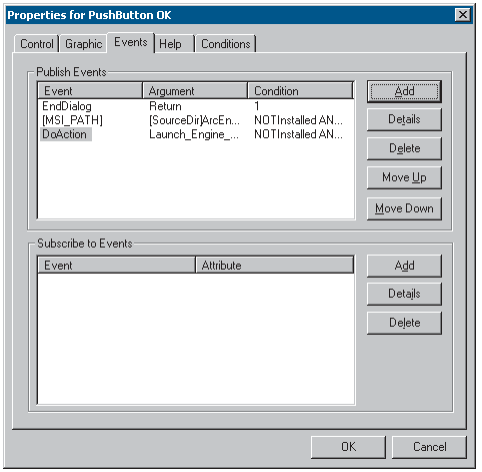
1. Create Properties MSI_PATH and ArcEngineExists and initialize them to 1 in the Property table.
2. Perform a system check for ArcGIS Engine Runtime at the beginning of your setup program. The system check should search for the RealVersion registry key and set the property ArcEngineExists to True if the registry value returned is 9.0 (for applications built on 9.0) or 9.1 (for applications built on 9.1).

   ```
   HKEY_LOCAL_MACHINE\Software\ESRI\ArcGIS Engine Runtime
   ```

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. Leave Command Line blank, as setup.exe is being launched.

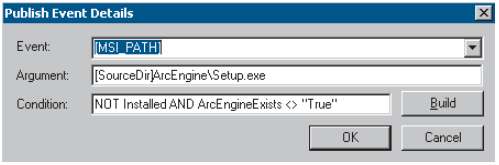


4. In the Exit dialog box of your application's setup program, add two actions behind the Finish button control.

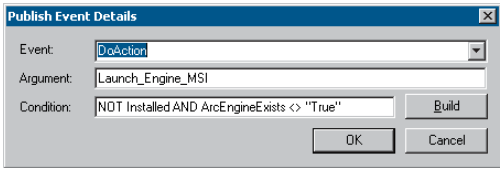


In the example on the right, MSI_PATH will change depending on where the ArcGIS Engine Runtime setup is located on the media. In this example, Setup.msi is located on the CD in a folder named ArcEngine.

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.



The second action calls the Launch_Engine_MSI that you previously created.



Both of these custom actions should execute only if the property ArcEngineExists does not equal True.

Example 2 (ArcGIS Engine Runtime 9.0 or 9.1 and a service pack are required.)

In this example, ArcGIS Engine Runtime and a service pack are required by your application. This example results in two scenarios. Scenario 1 takes into account the users that do not have ArcGIS Engine Runtime and the service pack installed. Scenario 2 takes into account the users that have ArcGIS Engine Runtime installed but do not have the service pack installed. Both these scenarios will require the following on your CD:

1. An admin installation that has been patched with the service pack.
2. The required ArcGIS Engine Runtime service pack (msp file).

The custom actions for scenarios 1 and 2 (described in the steps below) can be combined in the Finish button of your application's setup program. The Finish button of your application's setup program can then deal with both of these scenarios.

For scenario 1, you will have to provide an admin installation on the CD that consists of the original .msi file that has been patched with the required service pack. When deploying ArcGIS Engine Runtime in this particular deployment method, you cannot launch the original setup.msi file followed by a .msp file (service pack file).

On your CD you must provide an admin installation of ArcGIS Engine Runtime with the service pack applied. This will create a patched setup.msi file that you can install on your user's machine. The patched setup.msi file will contain both the entire ArcGIS Engine Runtime files and those files updated by the service pack. This is required for those users that do not have ArcGIS Engine Runtime and the required service pack on their machine.

To create an admin installation:

During the admin installation of ArcGIS Engine Runtime, you will be able to specify where you want the files installed. An admin installation does not write to your registry; it just creates a new image of the files required for clients to install.

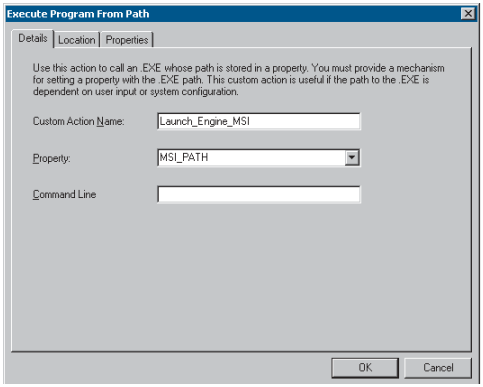
a. Install ArcGIS Engine Runtime using the following command line: msiexec /a <Engine Runtime CD>\setup.msi

b. Apply the service pack to this admin location using the command line (in the following example, Engine 9.0 service pack 1 is applied): msiexec /a <location of admin installation>\setup.msi /p <location of msp file>ArcGISEngine90sp1.msp REINSTALL=ALL REINSTALLMODE=omus

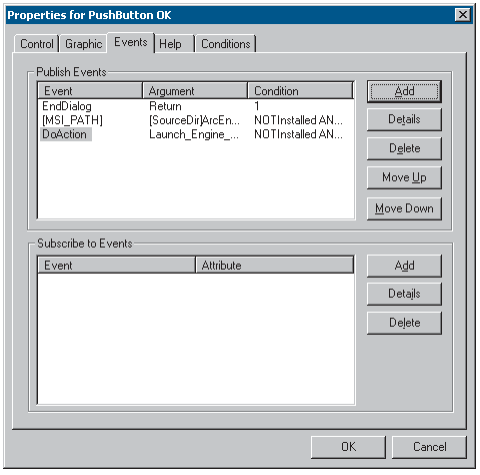
c. Copy the entire contents of your admin installation—which will now have the service pack applied—to your application's CD for redistribution.

Follow these steps to launch the setup.exe file for ArcGIS Engine Runtime when it's located in an ArcEngine folder on your application's media:

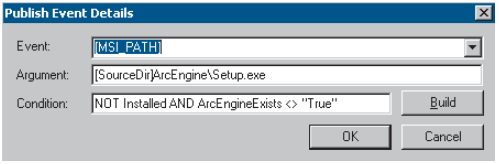
1. Create Properties MSI_PATH and ArcEngineExists and initialize them to 1 in the Property table.
2. Perform a system check for ArcGIS Engine Runtime at the beginning of your setup program. The system check should search for the RealVersion and SPNumber registry key and set the property ArcEngineExists to True if the registry values returned match your requirement.
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. Leave Command Line blank, as setup.exe is being launched.In the Exit dialog box of your application's setup program, add two actions behind the Finish button control.



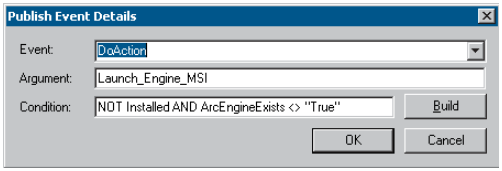
In the Exit dialog box of your application's setup program, add two actions behind the Finish button control.



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. The setup.exe file will be the admin installation with the required service pack applied.



The second action calls the Launch_Engine_MSI that you previously created.



Both these custom actions should execute only if the property ArcEngineExists does not equal True. ArcEngineExists should equal True only if ArcGIS Engine Runtime and the service pack are detected on the target machine.

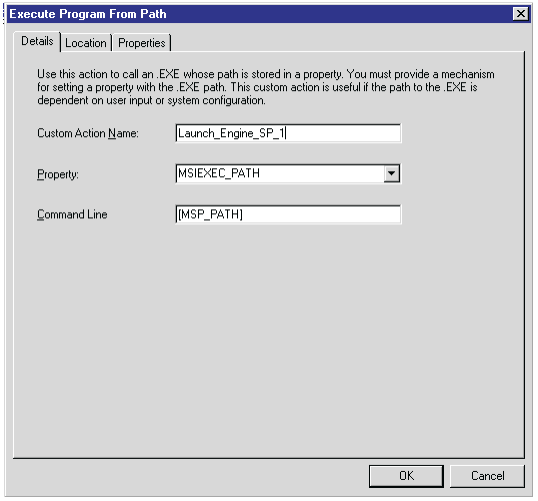
For scenario 2, you will have to provide the required ArcGIS Engine Runtime service pack (.msp file) on the CD. When deploying ArcGIS Engine Runtime in this particular deployment method, your user will have ArcGIS Engine Runtime already on the target machine, and only the service pack is required.

Follow these steps to launch the ArcGIS Engine Runtime service pack when it's on your application's media:

1. Create Properties MSP_PATH, MSIEXEC_PATH, and InstallSP and initialize them to 1 in the Property table.
2. Perform a system check for ArcGIS Engine Runtime and the service pack at the beginning of your setup program. The system check should search for the ArcGIS Engine Runtime RealVersion registry key and SPNumber registry key. The InstallSP property should be set to True if the RealVersion registry key value matches your requirement, but the SPNumber registry key value does not match your requirement. In this case, the machine has the ArcGIS Engine Runtime product installed but does not have the required service pack.
3. Create a custom action called Launch_Engine_SP_1. You can change the number to whichever version of the ArcGIS Engine Runtime service pack your application requires. This example is installing ArcGIS Engine Runtime service pack 1. This custom action will launch the service pack (.msp file) using the following syntax:

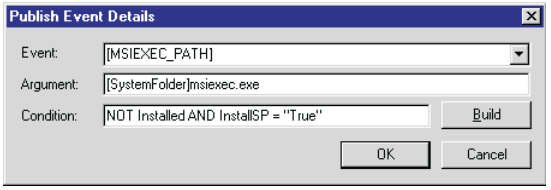
   msiexec.exe /p <location of service pack>\ArcGISEngine90sp1.msp /qn

4. Use the Execute Program from Path type of custom action. Set the Property in this custom action to MSIEXEC_PATH. The path to execute the program from is the path specified in the Property field, in this case MSIEXEC_PATH. The command line will be [MSP_PATH]. MSP_PATH is the path to the ArcGIS Engine Runtime service pack 1 .msp file that will be available on the media. Using the above syntax, MSIEXEC_PATH will be [SystemFolder]msiexec.exe. MSP_PATH will be /p <location of service pack>\ArcGISEngine90sp1.msp /qn. These two properties will be defined in the Exit dialog box's Finish button.

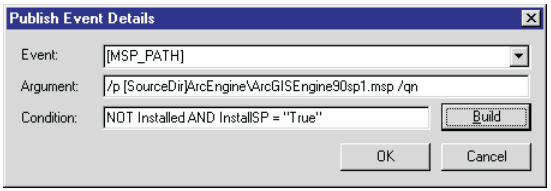


5. In the Exit dialog box of your application's setup program, add three actions behind the Finish button control.

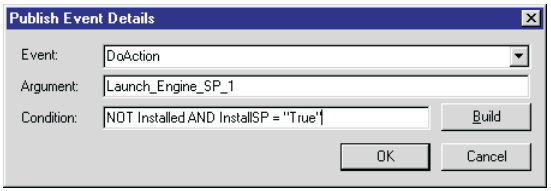
6. The first action sets the MSIEXEC_PATH property to [SystemFolder]msiexec.exe.



7. The second action sets the MSP_PATH to /p [SourceDir]ArcEngine\ArcGISEngine90sp1.msp /qn. MSP_PATH will change depending on where the ArcGIS Engine Runtime service pack is located on the media. In this example, ArcGISEngine90sp1.msp is located on the CD in a folder named ArcEngine.



8. The third action calls the Launch_Engine_SP_1 that you previously created.



These three custom actions should execute only if the property LaunchSP does not equal True.

Launching the ArcGIS Engine Runtime installation within a scripted setup

A second way you can install ArcGIS Engine Runtime within your own application is via a scripted setup that utilizes the available command-line parameters. The example script below was created using the Wise InstallMaster setup authoring software. It includes the installation of ArcGIS Engine Runtime and service pack 1.

In the example on the right, 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. MSP_PATH will change depending on where the ArcGIS Engine Runtime service pack is located. In this example, the service pack is located on the CD.

```
Rem Set variable
Set variable MSI_PATH to <CDROM>\ArcEngine\setup.msi
Set variable MSP_PATH to <CDROM>\ArcGISEngine90sp1.msp
Rem launch ArcGIS Engine Runtime setup program silently – No UI
Execute %SYS32%\msiexec.exe /i %MSI_PATH% /qn (Wait)
Rem launch ArcGIS Engine Runtime service pack silently – No UI
Execute %SYS32%\msiexec.exe /p %MSP_PATH% /qn (Wait)
```

Launching the ArcGIS Engine Runtime installation within a batch file

Yet another way that the ArcGIS Engine Runtime setup can be combined with your application's installation program is within batch files. The following example again takes advantage of the available command-line parameters and implements them within a batch file rather than a scripted setup. This example includes the installation of ArcGIS Engine Runtime and service pack 1.

```
REM ############################
REM Set variables
SET MSI_PATH=<CDROM>\ArcEngine\Setup.msi
SET MSP_PATH=<CDROM>\ArcGISEngine90sp1.msp

REM ############################

REM Launch MSI Silently - NO UI
msiexec.exe /i "%MSI_PATH%" /qn
msiexec.exe /p "%MSP_PATH%" /qn

REM Launch MSI Silently - Reduced UI
msiexec.exe /i "%MSI_PATH%" /qb
msiexec.exe /p "%MSP_PATH%" /qb

REM Launch MSI Silently - No UI except for a modal dialog box displayed at
the end.
msiexec.exe /i "%MSI_PATH%" /qn+
msiexec.exe /p "%MSP_PATH%" /qn+
```

The following examples also show the use of a batch file; however, these illustrate the command-line parameters that should be used to install the ArcGIS Engine Runtime Java or .NET features, respectively:

JavaRuntime:

```
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
```

In the example on the right, if .NET Framework 1.1 is not detected on the machine, the DotNetRuntime feature will not be installed.

DotNetRuntime:

```
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
```

ARCGIS ENGINE RUNTIME START MENU SHORTCUTS

By default, a complete installation of ArcGIS Engine Runtime 9.1 or later will create the following shortcuts on the ArcGIS start menu:

- Software authorization
- Authorization summary

The following command-line parameters provide options to not display these shortcuts on the start menu.

SHORTCUTS = YES	This option will install the shortcuts.
SHORTCUTS = NO	This option will not install the shortcuts.

In the following example, ArcGIS Engine Runtime is installed silently without shortcuts:

msiexec.exe /i <location of setup>\setup.msi /qn SHORTCUTS=NO

UNINSTALLING ARCGIS ENGINE RUNTIME

These are some points to consider before uninstalling ArcGIS Engine Runtime:

- ArcGIS Engine Runtime should be uninstalled using the Control Panel, not by deleting files from disk.
- Do not uninstall ArcGIS Engine Runtime during the uninstallation of your developed application.
- You should recommend strongly to your users that they manually uninstall ArcGIS Engine Runtime only when they know there are no third-party applications using it.

The previous section discussed installation on Windows; this section discusses installation on Solaris and Linux.

ABOUT THE ARCGIS ENGINE RUNTIME INSTALLATION PROGRAM

The ArcGIS Engine Runtime installation program, or setup, was created using ZeroG InstallAnywhere technology. This technology creates multiplatform packages that install and configure software on ArcGIS Engine Runtime-supported platforms (Solaris and Linux). The packages are launched using a properties file. The properties file contains installation parameters and indicate a silent mode of installation.

The ArcGIS Engine Runtime setup consists of features. A feature is an individual portion of the application's total functionality that may be installed independently. The ArcGIS Engine Runtime setup consists of the following installation features:

The available ArcGIS Engine Runtime setup features are illustrated in the table on the right.

Feature	Description
ArcGIS Engine Runtime	ArcGIS Engine components
Java Runtime	ArcGIS Engine Java components

All ArcGIS Engine software-based applications depend on an installation of the ArcEngine feature. Applications built using the Java API for ArcGIS Engine require installation of the JavaRuntime feature in addition to the ArcEngine feature.

Windows is also supported. For details on its system requirements, see the previous section, 'Installing ArcGIS Engine Runtime on Windows'.

ArcGIS Engine Runtime is supported on Sun Solaris 8 and 9 and Red Hat Linux AS/ES 3.0.

For additional or updated information regarding ArcGIS Engine Runtime system requirements, visit *http://support.esri.com*.

INSTALLING ARCGIS ENGINE RUNTIME

As mentioned above, ArcGIS Engine-based applications require that ArcGIS Engine Runtime be installed on the end user's machine. Installation of the runtime can be handled in either of two ways:

You cannot redistribute individual ArcGIS Engine Runtime files; the installation mechanisms discussed in this appendix are the only means of installing ArcGIS Engine Runtime files.

1. Your end user runs the ArcGIS Engine Runtime setup directly from the CD.
2. You include the ArcGIS Engine Runtime setup within your own application's installation program.

The following sections document the general requirements for the installation of the runtime, no matter which of the two types of installation mechanism you choose, and the steps necessary to install the runtime successfully using your chosen mechanism.

General requirements

Irrespective of the installation method you chose for ArcGIS Engine Runtime, you should be aware of the following:

- **Older versions of ArcGIS software**—ArcGIS Engine Runtime cannot be installed on any machine that already contains versions of ArcGIS products prior to the version you will be installing. This includes ArcIMS with ArcMap Server, ArcGIS Engine Developer Kit, ArcReader, ArcGIS Server, and ArcGIS Engine Runtime.

- **Existing installation of ArcGIS Engine Runtime**—If the ArcGIS Engine Runtime setup is launched, and ArcGIS Engine Runtime already exists on the machine, a message is displayed informing the user that the product should be installed to the same installation directory as the existing ArcGIS Engine Runtime.
- **Installation location**—Each of the ArcGIS 9 products (ArcGIS Engine Runtime, ArcGIS Engine Developer Kit, ArcReader, and ArcGIS Server) install to the same installation directory. The first ArcGIS 9 product installed determines the installation location for all subsequent ArcGIS 9 products. For example, if ArcReader is installed to /disk1/ArcGIS9, the installation location for all ArcGIS 9 products is /disk1/ArcGIS9/ArcGIS. If ArcGIS Engine Runtime is installed after ArcReader, the opportunity to browse to an installation location is not be provided. The initial ArcReader installation has predetermined the installation location for all future ArcGIS 9 products. Therefore, in this example, ArcGIS Engine Runtime will also be installed to /disk1/ArcGIS9/ArcGIS.

If the ArcMap Server component of ArcIMS 9 is already installed, ArcGIS 9 products will default to the ArcIMS installation location.

- **Disk space**—If you run out of disk space while installing an ArcGIS 9 product, you must uninstall all ArcGIS 9 products (listed above) and reinstall them to a location where more disk space is available. ArcGIS 9 products (excluding ArcSDE, ArcIMS, and ArcInfo Workstation) cannot be installed to different locations.

End user installs ArcGIS Engine Runtime on their system

In this case, your user installs the ArcGIS Engine Runtime either directly from the CD ESRI provided with the ArcGIS Engine product or from a CD that you created by copying the contents of the ArcGIS Engine Runtime CD image to it.

For additional ArcGIS Engine Runtime CDs, contact ESRI Customer Service at www.esri.com, *or in the USA call 888-377-4575, or contact your local ESRI regional office.*

If you create a custom CD that contains your own content as well as the ArcGIS Engine Runtime setup, the ArcGIS Engine Runtime directory structure must remain intact.

You cannot redistribute individual ArcGIS Engine Runtime files; the installation mechanisms discussed in this appendix are the only means of installing ArcGIS Engine Runtime files.

For example, the ArcGIS Engine Runtime CD image is as follows:

```
<CD_ROOT>
  ArcGISEngineRT/
  Setup
  support/
```

If you repackage the ArcGIS Engine Runtime CD image, your image may look similar to this:

```
<CD_ROOT>
  ThirdPartyStuff/
  MoreThirdPartyStuff/
  SomeArbitraryFolderName/
    ArcGISEngineRT/
     Setup
     support/
```

Whether you create a custom CD or provide your user with the ArcGIS Engine Runtime CD furnished by ESRI, you should distribute the following steps or similar instructions to your users:

1. Check whether ArcGIS Engine Runtime and the optional Java feature (if needed) have already been installed to the target machine by doing the following:

   a. Open the $HOME/ESRI.properties.<machinename> file.
   If the file exists, continue to Step 1b. If not, the ArcGIS Engine Runtime has not been installed; continue to Step 2.

   b. Check for the following property:

   `Z_ArcGISEngineRT_INSTALL_DIR`

   If it is listed, then ArcGIS Engine Runtime is installed on the target machine and you may not need to launch the ArcGIS Engine Runtime setup. Continue to Step 1c if your application requires the Java feature; otherwise, close the file and proceed to the installation steps for <your application name here> if needed.

   If the property is not listed, the ArcGIS Engine Runtime has not been installed. Continue to Step 2.

   c. Navigate to the installation location indicated by the Z_ArcGISEngineRT_INSTALL_DIR property. If the /java/arcgis_system.jar file is present, the ArcGIS Engine Runtime Java feature is installed on the machine. Proceed to the installation steps for <your application name here> if needed.

   If the file is not present, the JavaRuntime feature for ArcGIS Engine Runtime has not been installed. Continue to Step 2.

2. Confirm that any currently installed ArcGIS products are the same version as the version you are installing. This includes ArcIMS with ArcMap Server, ArcGIS Engine Developer Kit, ArcReader, ArcGIS Engine Runtime, and ArcGIS Server. To check if any of these products are installed on the machine and verify the version number:

   a. Open the $HOME/ESRI.properties.<machinename> file.

   b. Look for the Z_REAL_VERSION property to determine the versions of ArcGIS products installed on the machine.

   c. If the versions displayed are the same, proceed to Step 3. If the versions displayed are not the same, the products need to be removed before installing ArcGIS Engine Runtime. Do not proceed to Step 3 until these products have been uninstalled.

3. Launch the ArcGIS Engine Runtime installation program by executing the Setup file.

4. Click Next on the Welcome dialog box, review and accept the license agreement, then click Next to proceed with the installation.

If the ArcGIS Engine Runtime setup is launched, and ArcGIS Engine Runtime already exists on the machine, the installation dialog boxes will display a message informing you that the product will be installed to the same installation directory as the existing ArcGIS Engine Runtime.

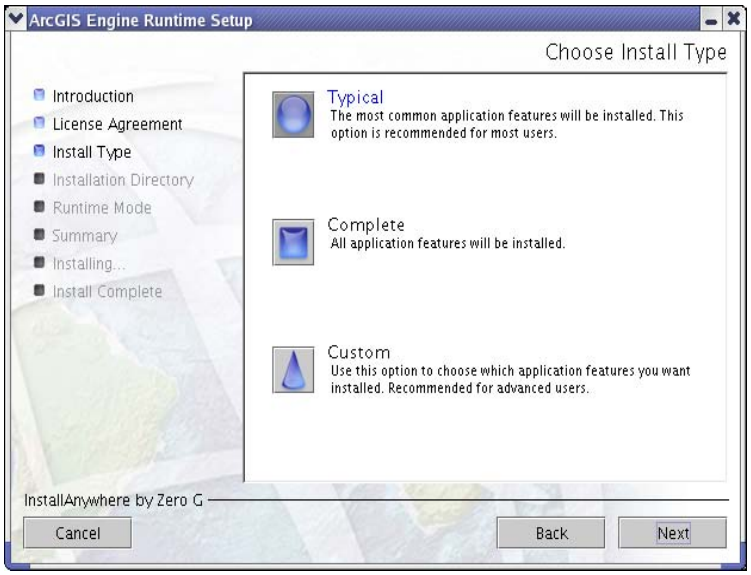
The Choose Install Type panel of the ArcGIS Engine Runtime installation wizard is illustrated on the right. The possible types are discussed below.

5. Choose the installation type and click Next to continue.

A Typical installation includes the ArcEngine feature only.

When Complete is selected all available ArcGIS Engine Runtime features are installed including the JavaRuntime.

If your ArcGIS Engine-based application utilizes Java, your user can select Custom and manually choose the JavaRuntime feature. As noted above a Complete installation also installs the JavaRuntime feature.



If the ArcGIS Engine software-based application to be installed requires Java, choose Custom to manually select the JavaRuntime feature or choose Complete to install all ArcGIS Engine Runtime installation features.

6. Follow any further instructions of the ArcGIS Engine Runtime installation wizard until it indicates that the installation is complete. If you installed, and intend to use, the Java feature, proceed to Step 7. If not, the installation of ArcGIS Engine Runtime is now complete. Proceed to the installation steps for <your application name here> if needed.

7. The JavaRuntime feature of ArcGIS Engine Runtime requires some additional postinstallation steps to be performed. Helper scripts, init_engine.xxx, are provided for this step.

   • If you use C-shell, source init_engine.csh.
   • If you use bash or bourne shell, source init_engine.sh.

The installation of ArcGIS Engine Runtime is now complete. Proceed to the installation steps for <your application name here> if needed.

ArcGIS Engine Runtime setup is included in your application's installation program

The second way to distribute the ArcGIS Engine Runtime is for you to incorporate its setup into your own ArcGIS Engine software-based application's installation program.

This is possible because ArcGIS Engine Runtime can be installed without a graphical user interface by triggering a silent install using command-line parameters in conjunction with a properties file. The command-line parameters trigger the silent install and specify the path to the properties file, while the properties

file itself establishes the necessary installation parameters. The following tables illustrate the required command-line parameters and the necessary properties in the file.

This table shows command-line parameters necessary to trigger a silent install of ArcGIS Engine Runtime.

Command-line parameters	Description
-s	Triggers silent (no user interaction) installation
/path/to/propertiesfile	Location of the property file on the CD image

This table shows properties that must be stored in a file to install ArcGIS Engine Runtime without user interaction.

Property names must be uppercase, as shown in this table.

Property	Value	Notes
INSTALLER_UI	SILENT	Required property. Instructs InstallAnywhere to operate in silent mode. SILENT must be in uppercase.
INSTALL_DIR	some/path/on/disk	Required property. Can be any valid disk location.
MWRT_MODE	Professional Enterprise	Required property. Can be any valid disk location.
INSTALL_TYPE	Complete Typical	Required property. A Typical install will include the ArcGIS Engine installation feature only. A Complete install will include ArcGIS Engine and Java installation features. No other level of detail can be specified.

A completed property file would be similar to the following:

```
INSTALLER_UI=SILENT
INSTALL_DIR=/disk1/myproduct
MWRT_MODE=Enterprise
INSTALL_TYPE=Typical
```

Your application setup program should not launch the ArcGIS Engine Runtime setup if it is already installed on the machine.

Your setup should follow these steps:

1. Before launching the ArcGIS Engine Runtime setup, you need to determine if ArcGIS Engine Runtime and the optional Java feature (if needed) already exist on the target machine.

   a. Open the $HOME/ESRI.properties.<machinename> file.
   If the file exists, continue to Step 1b. If not, the ArcGIS Engine Runtime has not been installed; continue to Step 2.

   b. For ArcGIS Engine Runtime, check for the following property:

   ```
   Z_ArcGISEngineRT_INSTALL_DIR
   ```

   If it is listed, then ArcGIS Engine Runtime is installed on the target machine and you may not need to launch the ArcGIS Engine Runtime setup. Continue to Step 1c if your application requires the Java feature; otherwise, close the file and proceed to the installation steps for <your application name here> if needed.

   If the property is not listed, the ArcGIS Engine Runtime has not been installed. Continue to Step 2.

   c. Navigate to the installation location indicated by the Z_ArcGISEngineRT_INSTALL_DIR property. If the /java/arcgis_system.jar file is present, the ArcGIS Engine Runtime Java feature is installed on the machine. Proceed to the installation steps for <your application name here> if needed.

If the setup is launched, and ArcGIS Engine Runtime already exists on the machine, the setup will execute again. If the setup is executed silently, the product will automatically be installed to the same installation directory as the existing ArcGIS Engine Runtime setup.

If the file is not present, the JavaRuntime feature for ArcGIS Engine Runtime has not been installed. Continue to Step 2.

2. Launch the ArcGIS Engine Runtime installation program by executing the following statement to launch the ArcGIS Engine Runtime setup silently:

   ```
   <location of ArcGISEngineRT image>/Setup –s /path/to/propertiesfile
   ```

   Any errors experienced during the silent install will be captured in a log file. It is advisable that you interact with the log file to determine if errors have occurred. The log file will be located in the user's $HOME directory:

   ```
   $HOME/ArcGISEngineRT_InstallLog.log
   ```

   'Installation log files', at the end of this section, provides some examples of the log files.

The following sections provide some examples of incorporating the ArcGIS Engine Runtime setup within your application's installation program.

Example A—Installing ArcGIS Engine Runtime using a script

The following is a sample install script that includes an initial check for ArcGIS Engine Runtime and the Java installation feature, launches the setup silently, and checks the log file for errors during installation.

```
#!/bin/sh

#
# check to see if ArcGIS Engine Runtime is already installed...
#
machineName='uname -n'
esriPropFile=$HOME/ESRI.properties.$machineName
if [ -f $esriPropFile ]
then
  # query prop file for the installation location of ArcGIS Engine
Runtime...
  runtimeHome='cat $esriPropFile | grep Z_ArcGISEngineRT_INSTALL_DIR'
  if [  "$runtimeHome" = "" ]
  then
    echo "ArcGIS Engine Runtime is NOT installed!"
  else
    echo "ArcGIS Engine Runtime IS installed!"

     #
    # check to see if the Java components of ArcGIS Engine Runtime are
installed...
     #
    engineHome='cat $esriPropFile | grep Z_ENGINE_HOME | cut -d= -f2'
    if [ -f $engineHome/java/arcgis_system.jar ]
     then
       echo "ArcGIS Engine Runtime Java components ARE installed!"
     else
       echo "ArcGIS Engine Runtime Java components ARE NOT installed!"
     fi
  fi
else
```

```
  echo "ArcGIS Engine Runtime is NOT installed!"
fi
#
# launch the ArcGIS Engine Runtime setup silently...
#
sh <location of ArcGISEngineRT image>/Setup -s /path/to/propertiesfile
```

Any errors experienced during the silent install will be captured in a log file. The log file will be located in the user's $HOME directory:

```
$HOME/ArcGISEngineRT_InstallLog.log
```

```
#
# check for errors in the ArcGIS Engine Runtime setup...
#
fatalErrors='cat $HOME/ArcGISEngineRT_InstallLog.log | grep " FATAL ERRORS"'
nonfatalErrors='cat $HOME/ArcGISEngineRT_InstallLog.log | grep "NONFATAL
ERRORS"'
warnings='cat $HOME/ArcGISEngineRT_InstallLog.log | grep "WARNINGS"'
if [ "`echo $fatalErrors | cut -d\" \" -f1`" != "0" ] ||
  [ "`echo $nonfatalErrors | cut -d\" \" -f1`" != "0" ] ||
  [ "`echo $warnings | cut -d\" \" -f1`" != "0" ]
then
  echo "ArcGIS Engine installed WITH ERRORS!"
  echo "Please view the log file for details."
  echo "  Log File: $HOME/ArcGISEngineRT_InstallLog.log"
else
  echo "ArcGIS Engine installed without error!"
fi
```

Example B—Installing ArcGIS Engine Runtime using ZeroG InstallAnywhere

The following is an example using ZeroG InstallAnywhere to launch the ArcGIS Engine Runtime setup at some point within your application's setup (if created using InstallAnywhere). The example includes a check for the presence of ArcGIS Engine Runtime and the Java installation feature.

1. To perform a system check for ArcGIS Engine Runtime, add an "Execute Script/Batch File" action with the following parameters:
   a. "Suspend execution until operation is complete" checked ON.
   b. "Store process's stdout in:" = a variable name of your choice (for example, $RT_INSTALLED_OUT$).
   c. Script:

```
#!/bin/sh

machineName='uname -n'
esriPropFile=$DOLLAR$HOME/ESRI.properties.$DOLLAR$machineName
if [ -f $DOLLAR$esriPropFile ]
then
  runtimeHome='cat $DOLLAR$esriPropFile | grep
    Z_ArcGISEngineRT_INSTALL_DIR'
  if [ "$DOLLAR$runtimeHome" != "" ]
  then
    echo "installed"
  fi
```

```
fi
exit 0
```

When this command execution is complete, if ArcGIS Engine Runtime is installed, the value of $RT_INSTALLED_OUT$ will be "installed".

2. To perform a system check for the Java installation feature, add an "Execute Script/Batch File" action with the following parameters:

   a. "Suspend execution until operation is complete" checked ON.

   b. "Store process's stdout in:" = a variable name of your choice (for example, $JAVA_INSTALLED_OUT$).

   c. Script:

```
#!/bin/sh

machineName='uname -n'
esriPropFile=$DOLLAR$HOME/ESRI.properties.$DOLLAR$machineName
if [ -f $DOLLAR$esriPropFile ]
then
  runtimeHome='cat $DOLLAR$esriPropFile | grep
    Z_ArcGISEngineRT_INSTALL_DIR'
  if [ "$DOLLAR$runtimeHome" != "" ]
  then
       engineHome='cat $DOLLAR$esriPropFile | grep Z_ENGINE_HOME |
         cut -d= -f2'
       if [ -f $DOLLAR$engineHome/java/arcgis_system.jar ]
       then
         echo "installed"
       fi
    fi
  fi
fi

exit 0
```

   When this command execution is complete, if the ArcGIS Engine Runtime Java installation feature is installed, the value of $JAVA_INSTALLED_OUT$ will be "installed".

3. To launch the ArcGIS Engine Runtime setup within your application's InstallAnywhere setup, add an "Execute Command" action with the following parameters:

   a. "Suspend execution until operation is complete" checked ON.

   b. "Command line" = <location of ArcGISEngineRT image>/Setup –s /path/to/properties/file

4. To execute this command conditionally, based on the system check for ArcGIS Engine Runtime (as described in Step 1 above), add a "Compare InstallAnywhere Variable" rule with the following parameters:

   a. "Operand 1" = $RT_INSTALLED_OUT$

   b. Operation = "does not equal"

c. "Operand 2" = "installed" (without the quotes)

You can add the Execute Command action at any point within the setup (Pre-Install, Install, or Post-Install). If your application needs to be installed on top of ArcGIS Engine Runtime, then you will need to install ArcGIS Engine Runtime in the Pre-Install step of an InstallAnywhere setup.

INSTALLING ARCGIS ENGINE RUNTIME SERVICE PACKS

The ArcGIS Engine Runtime service packs need to be installed by the end user. The service packs can be obtained from *http:\\support.esri.com*.

If you require a service pack for your application, you can advise your users to download the service packs from *http:\\support.esri.com*, or you can copy the service pack (.tar file) to your CD, along with ArcGIS Engine Runtime for redistribution.

INSTALLATION LOG FILES

It is advisable that you interact with the log file to determine if errors have occurred during installation.

Example A—without errors

Below is an example of an error-free log file.

```
Install Begin: Wed Aug 27 14:39:57 PDT 2003
Install End:   Wed Aug 27 14:44:25 PDT 2003

Created with Zero G's InstallAnywhere 5.5.1 Enterprise Build 2032


Summary
___

Installation: Successful.

30 SUCCESSES
0 WARNINGS
0 NONFATAL ERRORS
0 FATAL ERRORS

Action Notes:

None.

Install Log Detail:

Install Action:         InstallAnywhere Variable
                       Status: SUCCESSFUL

Install Action:         InstallAnywhere Variable
                       Status: SUCCESSFUL

Install Action:         InstallAnywhere Variable
                       Status: SUCCESSFUL
```

```
Install Action:         Evaluating System Requirements...
                        Status: SUCCESSFUL

Install Action:         Verifying Install Properties
                        Status: SUCCESSFUL

Install Action:         Reading ESRI properties file...
                        Status: SUCCESSFUL

Install Action:         Checking for Previous Installation...
                        Status: SUCCESSFUL

Install Action:
                        Status: SUCCESSFUL


Install Action:         Calculating Disk Cost...
                        Status: SUCCESSFUL

Install Action:         InstallAnywhere Variable
                        Status: SUCCESSFUL

Install Action:         Determining Install Dir
                        Status: SUCCESSFUL

Install Action:         InstallAnywhere Variable
                        Status: SUCCESSFUL

Install Action:         Load Local Registry
                        Status: SUCCESSFUL

Install Directory:      /mozart1/silent/arcgis/
                        Status: SUCCESSFUL

Install Merge Module:
$MODULUS_MM_DIR$\Engine_Build_Output\Merge_Modules\Engine.iam.zip
                        Status: SUCCESSFUL
                       Additional Notes: NOTE - Installing:
                        SubInstaller1.zip

Custom Action:          com.esri.ia.actions.IncrementRefCount_Engine
                        Status: SUCCESSFUL

Custom Action:          com.esri.ia.actions.ReadESRIPropFile
                        Status: SUCCESSFUL

Install Directory:      /mozart1/silent/arcgis/.Setup/
                        Status: SUCCESSFUL

Install Directory:      /mozart1/silent/arcgis/.Setup/UninstallArcGISEngineRT/
                        Status: SUCCESSFUL
```

```
                    Additional Notes: NOTE - Directory already existed

Install Uninstaller:     ArcGISEngineRT (Install All Uninstaller Components)
                         Status: SUCCESSFUL


Install File:           /mozart1/silent/arcgis/.Setup/UninstallArcGISEngineRT/
                        Uninstall_ArcGISEngineRT.lax
                         Status: SUCCESSFUL


Install Uninstaller:     ArcGISEngineRT (Install All Uninstaller Components)
                         Status: SUCCESSFUL

Modify Text File - Single File:  New File /mozart1/silent/arcgis/.Setup/
                                 ArcGISEngineRT.reg
                         Status: SUCCESSFUL

Modify Text File - Single File:  New File /mozart1/silent/arcgis/.Setup/
                                 registerArcGISEngineRT
                         Status: SUCCESSFUL

Modify Text File - Single File:  New File /mozart1/silent/arcgis/
                                 uninstallArcGISEngineRT
                         Status: SUCCESSFUL

Execute Script/Batch file:       Registering ArcGIS Engine Runtime
                         Status: SUCCESSFUL

Custom Action:          com.esri.ia.actions.RegisterProductInstallDir
                         Status: SUCCESSFUL


Install Action:          Checking for Merge Module Error...
                         Status: SUCCESSFUL
```

Example B—with errors

Below is an example of a log file with errors, signifying an unsuccessful installation.

```
Install Begin: Wed Aug 27 14:33:11 PDT 2003
Install End:   Wed Aug 27 14:33:11 PDT 2003

Created with Zero G's InstallAnywhere 5.5.1 Enterprise Build 2032


Summary
---

Installation: Unsuccessful.

4 SUCCESSES
```

```
0 WARNINGS
0 NONFATAL ERRORS
1 FATAL ERRORS

Action Notes:

None.

Install Log Detail:

Install Action:         InstallAnywhere Variable
                       Status: SUCCESSFUL


Install Action:         InstallAnywhere Variable
                       Status: SUCCESSFUL

Install Action:         InstallAnywhere Variable
                       Status: SUCCESSFUL

Install Action:         Evaluating System Requirements...
                       Status: SUCCESSFUL

VerifyInstallProperties
                         Status: FATAL ERROR
                      Additional Notes: FATAL ERROR - MainWin runtime
                       mode not specified
```

UNINSTALLING ARCGIS ENGINE RUNTIME

Uninstall by executing the uninstallArcGISEngineRT file found in the install directory.

These are some points to consider before uninstalling ArcGIS Engine Runtime:

- ArcGIS Engine Runtime should be uninstalled using the uninstaller provided, not by deleting files from disk.
- Do not uninstall ArcGIS Engine Runtime during the uninstallation of your developed application.
- You should recommend strongly to your users that they manually uninstall ArcGIS Engine Runtime only when they know there are no third-party applications using it.

E

Glossary

The following is a glossary of common terms used throughout this book. While it is not meant to be a comprehensive list, it should provide you with a quick reference to ArcGIS Engine software-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 the text and HTML tags that standard Web documents contain, 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, and Java.

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 seven-bit binary number (a string of seven 1s and 0s). ASCII defines 128 possible characters.

ASP See Active Server Pages.

ASP.NET A programming framework built on the Common Language Runtime 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.

association Represents relationships between classes. They have defined multiplicities at both ends.

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 executed by the computer.

binding The process of matching the location of a function given a pointer to an object.

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 referring 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, life cycle 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.

composition A stronger form of aggregation in which objects from the "whole" class control the lifetime of objects from the "part" class.

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 postinstallation 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 are running. Container processes run on SOC machines and are started and shut down by the Server Object Manager (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 A command that puts the address standardizer into debugging mode.

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 nonpooled 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 A type of file that stores shared code to be used by multiple programs (a code library). Programs access the shared code by linking to the .dll file when they run, a process referred to as dynamic linking. A .dll file must be registered for other programs to locate it. See also register.

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, ArcGIS Network Analyst, 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

composed 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 occurred 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. See also out-of-process.

instantiation Specifies that one object from one class has a method with which it creates an object from another class.

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, a 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.

marshalling 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.

n-ary association Specifies that more than two classes are associated. A diamond is placed at the intersection of the association branches.

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 local or wide area network (LAN or WAN).

object In object-oriented programming, an instance of the data structure and behavior defined by 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, Inc., 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 representation 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, marshalling 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 postinstallation 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 object 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 9, 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 onscreen 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.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X