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.

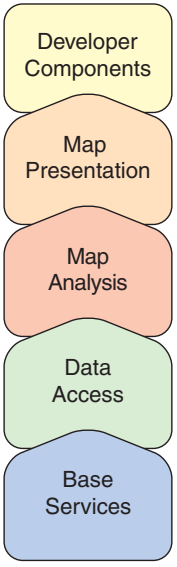
The ArcGIS software architecture supports a number of products, each with its unique set of requirements. ArcObjects, the components that make up ArcGIS, are designed and built to support this. This chapter introduces ArcObjects.

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'.

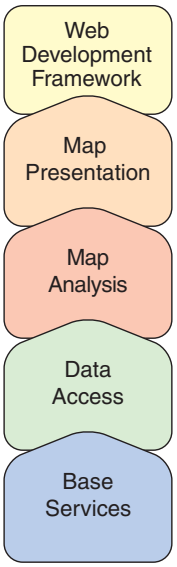
As stated, the language chosen to develop ArcObjects was C++; in addition to this language, ArcObjects makes use of the Microsoft Component Object Model. COM is often thought of as simply specifying how objects are implemented and built in memory and how these objects communicate with one another. While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

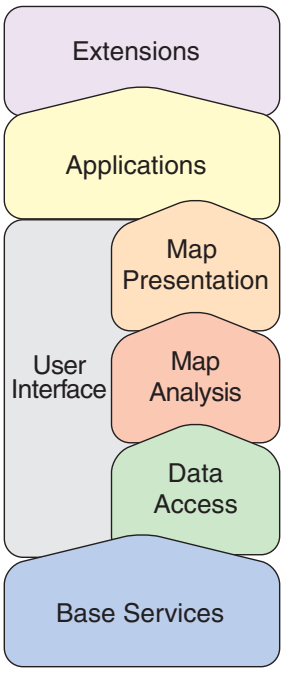


ArcGIS Engine

- 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.

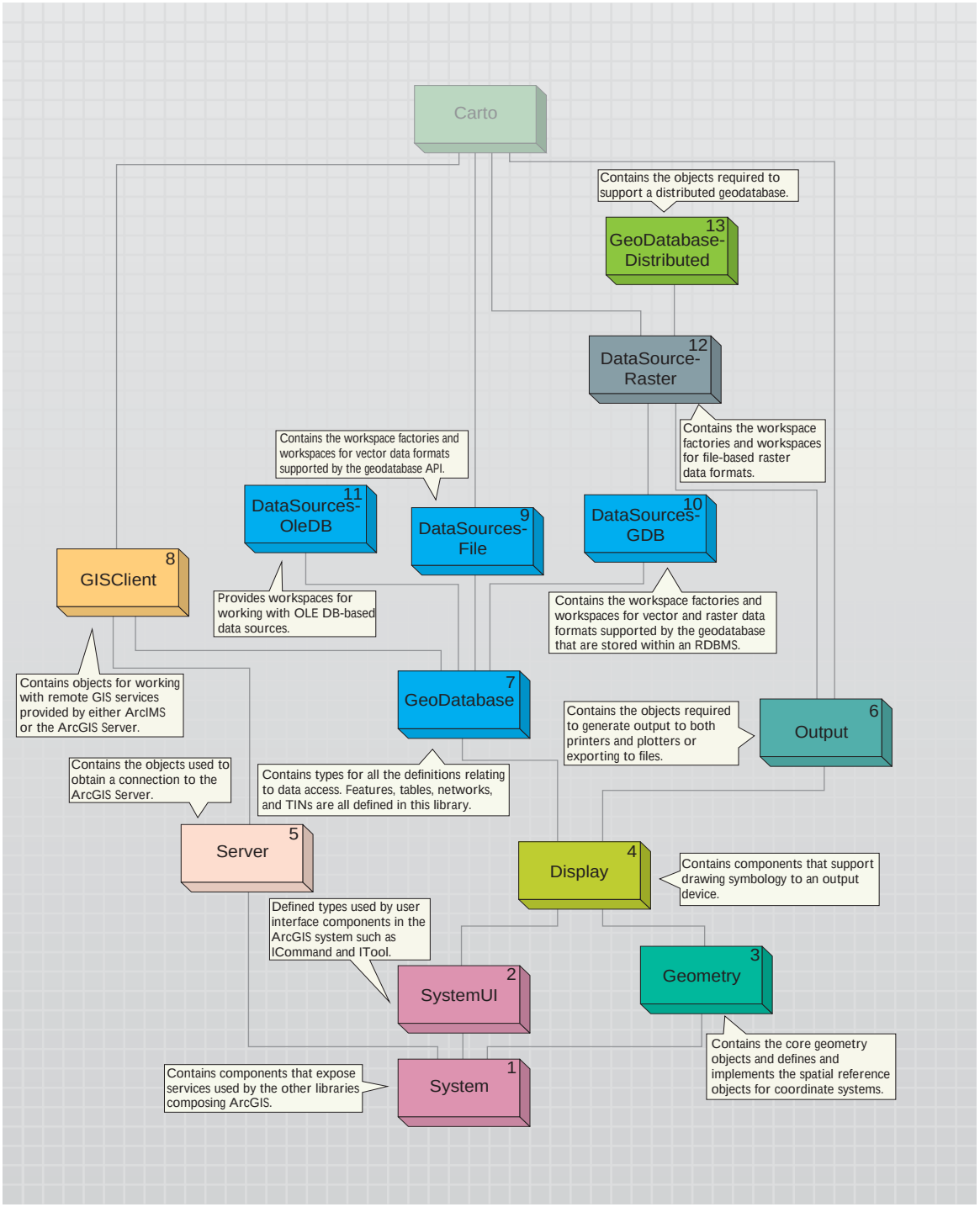
Type	Description
Boolean	Data item that can have the value True or False.
unsigned char	8-bit unsigned data item.
double	64-bit IEEE floating-point number.
float	32-bit IEEE floating-point number.
int	Signed integer, whose size is system dependent.
long	32-bit signed integer.
short	16-bit signed integer.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

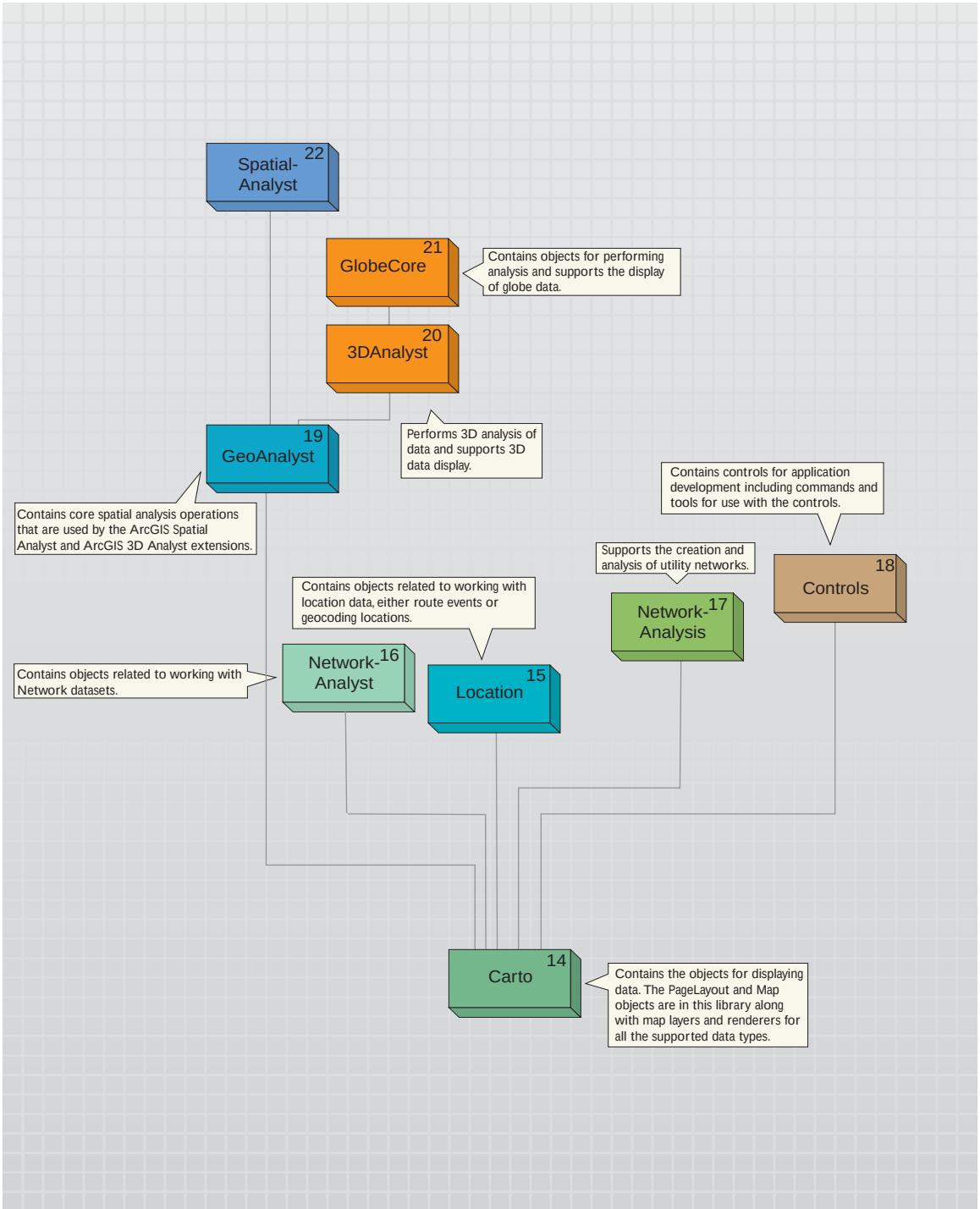
The DataSourcesRaster library contains the implementation of the GeoDatabase API for the raster data sources. These data sources include relational database management systems supported by ArcSDE—IBM DB2, Informix, Microsoft SQL Server, and Oracle—along with supported Raster Data Objects (RDO) raster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


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

SPATIAL ANALYST

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.