ADO: ActiveX Data Objects

ADO (ActiveX Data Objects) is Microsoft’s universal, language-independent solution for applications to easily access data in multiple formats located on different platforms. Data may be contained in email messages, Access databases, Word documents, Oracle SQL databases, and other formats. Developers need to access these sources without spending extra time learning every last detail about each format. ADO saves you time by providing a consistent interface for language-independent development, so that you can concentrate on the content and quality of your application, rather than on the nitty-gritty of specific data formats.

ADO: ActiveX Data Objects provides you with a solid introduction and reference to ADO, including:

- Chapters dedicated to the Connection, Recordset, Field, and Command objects and the Properties collection
- A complete, detailed reference listing every ADO object, method, property, and event, in convenient alphabetical order
- Chapters on ADO architecture, data shaping, and the ADO Event Model
- An appendix containing enumeration tables used by ADO objects and collections, listed alphabetically
- Brief introductions to RDS, ADO.NET, and SQL

ADO: ActiveX Data Objects provides essential information on ADO through Version 2.6, including practical code examples along the way. This book is an indispensable guide and comprehensive reference to ADO objects, collections, methods, and properties for developers who want to get up to speed and master ADO.

Jason T. Roff is the author of three database-development books, including ADO: ActiveX Data Objects. He specializes in Visual Basic, ASP, and SQL Server development and architecture and has experience working with C++ and Assembly on everything ranging from a Commodore to a Unix box.
ADO (ActiveX Data Objects) is Microsoft's universal, language-independent solution for applications to easily access data in multiple formats located on different platforms. Data may be contained in email messages, Access databases, Word documents, Oracle SQL databases, and other formats. Developers need to access these sources without spending extra time learning every last detail about each format. ADO saves you time by providing a consistent interface for language-independent development, so that you can concentrate on the content and quality of your application, rather than on the nitty-gritty of specific data formats.

**ADO: ActiveX Data Objects** provides you with a solid introduction and reference to ADO, including:

- Chapters dedicated to the Connection, Recordset, Field, and Command objects and the Properties collection
- A complete, detailed reference listing every ADO object, method, property, and event, in convenient alphabetical order
- Chapters on ADO architecture, data shaping, and the ADO Event Model
- An appendix containing enumeration tables used by ADO objects and collections, listed alphabetically
- Brief introductions to RDS, ADO.NET, and SQL

**ADO: ActiveX Data Objects** provides essential information on ADO through Version 2.6, including practical code examples along the way. This book is an indispensable guide and comprehensive reference to ADO objects, collections, methods, and properties for developers who want to get up to speed and master ADO.

Jason T. Roff is the author of three database-development books, including **ADO: ActiveX Data Objects**. He specializes in Visual Basic, ASP, and SQL Server development and architecture and has experience working with C++ and Assembly on everything ranging from a Commodore to a Unix box.
ADO: ActiveX Data Objects
ADO: ActiveX Data Objects

Jason T. Roff
# Table of Contents

**Preface** ........................................................................................................................................... ix

1. **Learning ADO** ......................................................................................................................... 1

   1. **Introduction to ADO** ........................................................................................................... 3
      ADO in Context: Universal Data Access ................................................................. 3
      ADO and COM: Language Independence .......................................................... 13
      When to Use ADO ................................................................................................. 15
      Summary ............................................................................................................... 18

   2. **The ADO Architecture** ...................................................................................................... 19
      An Overview of the ADO Architecture ............................................................. 19
      ADO Components ............................................................................................... 21
      Summary ............................................................................................................... 29

   3. **Accessing ADO with Various Languages** ........................................................................ 31
      Accessing ADO with Visual Basic ........................................................................ 31
      Accessing ADO with Visual C++ ......................................................................... 36
      Accessing ADO with Visual J++ .......................................................................... 40
      Accessing ADO with VBScript ............................................................................ 43
      Accessing ADO with JScript ................................................................................ 47
      Summary ............................................................................................................... 50
4. *The Connection Object* ........................................................................ 51
   Opening and Closing a Connection: Implicit Versus Explicit ............... 51
   Configuring Connections ........................................................................ 54
   Choosing a Data Provider ........................................................................ 70
   Executing Commands ............................................................................. 73
   Managing Multiple Transactions ......................................................... 77
   Determining the Layout of Your Data Source ..................................... 85
   Summary ................................................................................................. 91

5. *The Recordset Object* ....................................................................... 92
   Cursors: Viewing a Recordset ............................................................... 92
   Working with Recordsets ....................................................................... 101
   Navigating a Recordset .......................................................................... 115
   Working with Records ........................................................................... 128
   Lock Types: Managing Access to a Recordset .................................... 135
   Summary ............................................................................................... 141

6. *Fields* .............................................................................................. 143
   The Fields Collection Object ............................................................... 143
   Field Specifics ....................................................................................... 145
   Determining Field Object Functionality ............................................ 155
   Summary ............................................................................................... 157

7. *The Command Object* .................................................................... 158
   Specifying Commands .......................................................................... 158
   Executing Commands ........................................................................... 163
   Parameters ............................................................................................ 167
   The Parameters Project ......................................................................... 172
   Asynchronous Execution ....................................................................... 177
   Summary ............................................................................................... 182

8. *The ADO Event Model* .................................................................. 183
   Introduction to Events ........................................................................... 183
   The ConnectionEvent Family ............................................................. 185
   The RecordsetEvent Family ................................................................. 199
   Canceling Operations ........................................................................... 212
   Turning Events Off ................................................................................ 213
   Summary ............................................................................................... 214
9. Data Shaping ................................................................. 216
   An Introduction to Data Shaping ........................................... 216
   The Microsoft Data Shaping Service ................................. 217
   Shaping Commands ........................................................ 218
   Example: Accessing Shaped Recordsets ............................ 227
   Summary ............................................................................ 235

10. Records and Streams .................................................. 236
    The Record Object .......................................................... 236
    The Stream Object .......................................................... 252
    Summary ............................................................................ 259

11. Remote Data Services ................................................ 261
    RDS Object Model .......................................................... 261
    An Example in RDS ........................................................ 272
    More Information About RDS ......................................... 274
    Summary ............................................................................ 274

12. The Microsoft .NET Framework and ADO.NET ............ 275
    The Microsoft .NET Framework .................................... 275
    ADO.NET ........................................................................... 277
    ADO.NET Features ......................................................... 278
    Summary ............................................................................ 280

II. Reference Section ....................................................... 281

13. ADO API Reference ..................................................... 283
    Finding the Reference Page ............................................. 283
    Using the Reference Pages .............................................. 288

III. Appendixes ............................................................... 459

A. Introduction to SQL ..................................................... 461

B. The Properties Collection ............................................ 481

C. ADO Errors ................................................................. 500
D. The ADO Data Control .............................................................. 516
E. Enumeration Tables ............................................................. 536
Index ....................................................................................... 585
This book is about ActiveX Data Objects (ADO), including Version 2.6, the latest release of ADO from Microsoft at the time of publication. In this Preface, I will first briefly introduce ADO and explain how the book is organized.

Introduction and Organization

This book is organized into three parts, as described in the following sections.

Part I: Learning ADO

ADO is Microsoft’s advanced universal data-access solution, consisting of an object model–based wrapper around OLE DB, which is a technology that allows data-access functionality to different types of data sources. This allows companies such as Oracle, Microsoft, and Sybase to develop what are called “data providers,” to do just that—provide data to the OLE DB technology. OLE DB technology can work with all kinds of data sources, including relational databases such as SQL Server or an email system such as Exchange. OLE DB and ADO can even deal with plain text files and Excel spreadsheets. Chapter 1, Introduction to ADO, and Chapter 2, The ADO Architecture, provide more information on ADO, related technologies, and the structure of key ADO components.

ADO adds a common programming interface to OLE DB, thus allowing developers to use existing skills with multiple languages. ADO can be used with virtually any development language that supports COM, such as Visual Basic, Visual C++, J++, JScript, and VBScript. Developing with ADO in each of these languages is discussed in Chapter 3, Accessing ADO with Various Languages. ADO was designed to encourage DAO and RDO developers to migrate to this new technology, without the burden of the many different objects of DAO and RDO.
ADO is a lightweight, disconnected object model, which means that it has few objects, as compared to DAO or RDO, and that the objects do not necessarily rely on each other. For instance, one of the most common objects of ADO is the Connection object (Chapter 4, *The Connection Object*). This object establishes a physical connection with a data source. But you don’t need it: the other objects of ADO, such as the Command object, which issues textual commands to the data source, and the Recordset object (Chapter 5, *The Recordset Object*), which is used to store a result set, can create their Connection objects internally if they need to. Of course they use some default options, and hence the advantage of creating your own Connection—more power and control over your data access.

The Fields Collection object represents, unsurprisingly, a collection of fields contained in every Recordset object. Chapter 6, *Fields*, explains the Fields Collection object, as well as the Field objects.

Another example of ADO disconnected object model is the Command object, covered in Chapter 7, *The Command Object*. The Command object issues commands such as SQL statements. You can actually issue statements through the Connection object if you don’t mind using the default values. In this case the Connection object creates its own Command object internally to get the job done.

Asynchronous operations are a very big selling feature with a data-access technology—and ADO definitely does not fall short in this category. With the ability to fire events when asynchronous operations are executing and when they complete, ADO offers much greater control of your data access than did previous data-access technologies such as DAO. In addition to asynchronous operations, events can be fired for transactions, connecting and disconnecting to a data source, as well as moving around a recordset and changing values within it. Events are covered in Chapter 8, *The ADO Event Model*.

One of the unique features of ADO is its ability to use the Data Shaping data provider, which allows you to write code that can store hierarchical data within a single Recordset object. It allows you to shape result sets into parent-child relationships, where a single field value can contain an entire child recordset. Data shaping is covered in Chapter 9, *Data Shaping*.

A newer functionality in ADO is the ability to connect to web resources with not only the Recordset object, which stores result sets, but with the Record object, which stores individual rows, and the Stream object, which represents the actual content of a resource, such as a file or a directory. Chapter 10, *Records and Streams*, explains these topics.

Remote Data Services (RDS) adds functionality to three-tier web applications. Chapter 11, *Remote Data Services*, provides an overview of RDS.
Chapter 12, *The Microsoft .NET Framework and ADO.NET*, offers a glimpse into the next generation of ADO and related technologies, in the form of ADO.NET and the .NET Framework and how they will interact with today’s ADO projects.

**Part II: Reference Section**

Part II consists of Chapter 13, *ADO API Reference*. For this chapter, I have compiled an exhaustive list of every object, method, property, event, and enumeration in an easy-to-use alphabetical reference. See also Appendix E.

**Part III: Appendixes**

Appendix A, *Introduction to SQL*, provides just that—an introduction to using SQL with the Microsoft Jet Engine SQL language, including record selection, data manipulation, and database modification.

In Appendix B, *The Properties Collection*, I explain the Properties collection, which exists within and provides information about ADO objects. ADO is a flexible framework that exposes the functionality of the data provider. Nothing guarantees what functionality a data provider will actually provide your application, but ADO does dictate the interface used for supported functionality. ADO has what it calls “dynamic properties,” which can be used to understand the functionality supported by the data provider and to set data provider specific properties that aren’t part of the ADO framework. This flexibility that ADO offers contributes to its longevity.

Appendix C, *ADO Errors*, lists trappable errors and data-provider errors, as well as methods for handling them.

Appendix D, *The ADO Data Control*, explains the ADO Data Control Property Pages and how to create connection strings with the Data Control property, including an example application.

The companion to the Chapter 13 reference is Appendix E, *Enumeration Tables*, which alphabetically lists enumerations used by ADO objects and collections.

**About the Book**

This book covers ActiveX Data Objects up to Version 2.6. It covers every class, method, property, and enumeration included with this release. This book has three sections; the first is a tutorial that explains how each of these components work, with examples in Visual Basic along the way. The second part of this book is a practical reference guide that allows you to quickly look up any component to see every piece of detailed information available for it. The third part of this book contains several appendixes providing related information, as well as reference tables.
Although this book includes small sections on Remote Data Objects (RDO), ADO.NET (from Microsoft’s .NET Framework), and SQL, it by no means attempts to cover these subjects to any degree of completeness.

**Audience**

While this book is intended for any person interested in learning about ADO, it is targeted more specifically to the experienced Visual Basic developer who understands the basic principles behind data access and manipulation. This book provides many introductions to secondary topics, including SQL (Appendix A), RDS (Chapter 11), and others, in order to help the less-experienced reader understand all facets of ADO in context.

This book assumes that you know how to develop in Visual Basic—or you at least understand how to read it. Knowledge of one of Microsoft’s early database technologies (DAO or RDO) is helpful, but not necessary.

**Conventions Used in This Book**

I use the following font conventions in this book:

*Italic* is used for:

- New terms where they are defined
- Internet addresses, such as domain names and URLs
- Pathnames, filenames, and program names

*Constant width* is used for:

- Code examples for Visual Basic, C++, Java, and other languages
- Specific names and keywords in Visual Basic programs, including method names, property names, variable names, enumeration names, constants, and class names

*Constant width italic* is occasionally used for placeholder items in code, replaceable by a specific item in your code.

**Comments and Questions**

I have tested and verified the information in this book to the best of my ability, but you may find that features have changed (or even that I have made mistakes!).
Please let me know about any errors you find, as well as your suggestions for future editions, by writing to:

O'Reilly & Associates, Inc.
101 Morris Street
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (fax)

There is a web page for this book, which lists errata, any plans for future editions, or any additional information. You can access this page at:

http://www.oreilly.com/catalog/ado/

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about books, conferences, software, Resource Centers, and the O'Reilly Network, see the O'Reilly web site at:

http://www.oreilly.com

Acknowledgments

The people I need to acknowledge the most are the good folk at O'Reilly & Associates, starting with Ron Petrusha, who put up with me while still insisting on a quality piece of work. John Osborn and Nancy Kotary brought it home. Thank you very much for your expertise, guidance, persistence, and understanding.

I need to thank the technical reviewers who—while they didn’t go easy on me—didn’t beat me up too bad, either. This includes Bob Beauchemin and Ben Willet’s MDAC team over at Microsoft: Steve Hoberecht, Rick Feinauer, and Irene Smith. I’d also like to thank the O’Reilly Production staff. Specifically, thanks to my Production Editors, Jeffrey Holcomb and Sarah Jane Shangraw, and the additional Production staff who worked on this book: Linley Dolby, Matt Hutchinson, and Claire Cloutier.

And last but not least, my wife, who put up with me working on this book, before we were married, after we were married, before she was pregnant, while she was pregnant, and after she gave birth to my son, Zachary—the real reason I finished this book. I love you both.
I

Learning ADO
In today’s computing environments, data exists in many formats, ranging from Access and SQL Server databases to Word documents, email messages, and many others. ADO, or ActiveX Data Objects, data-access technology simplifies use of data from multiple sources, thus freeing developers from learning data, vendor-specific API calls, and any other coding minutiae for each data format involved. With ADO, almost any data source becomes accessible in a consistent way for developers creating standalone applications, client/server applications, or ASP pages.

In this chapter, I define ADO in the historic and current context of Microsoft’s overall data-access strategy and related technologies.

ADO in Context: Universal Data Access

Microsoft’s philosophy behind ADO and a series of related technologies is Universal Data Access (UDA). UDA isn’t a tangible product or technology, but rather a strategy for attacking the problem of data access, whose goal is efficient and powerful data access, regardless of data source or development language. Moreover, this universal access is meant to eliminate the need to convert existing data from one proprietary format to another.

With this lofty goal in view, Microsoft developed a series of technologies, collectively known as Microsoft Data Access Components (MDAC), that allow developers to implement UDA. MDAC consists of the following four key pieces:

- ODBC (Open Database Connectivity)
- OLE DB (Object Linking and Embedding Databases)
• ADO (ActiveX Data Objects)
• RDS (Remote Data Service)

These components implement the UDA vision both individually and as a whole. To best understand ADO in context, you should have a basic understanding of each MDAC technology and its relationship to ADO.

**ODBC**

Open Database Connectivity, or ODBC, provides access to relational databases through a standard API, addressing the problem of native application—and platform-specific APIs and their lack of cross-application compatibility. ODBC's industry-standard architecture offers an interface to any Database Management System (DBMS), such as SQL Server or Oracle, that uses the standard ODBC API. The main drawbacks of ODBC are the amount of work required to develop with it and its restriction to SQL-based data sources.

Two COM components (Component Object Model—see “ADO and COM: Language Independence” later in this chapter) designed to help with ODBC complications are DAO and RDO, described briefly in later sections in this chapter.

**Jet/DAO**

With the release of Microsoft Access 1.1 in 1993, Microsoft introduced the Jet Database Engine, which worked with Access databases (Microsoft Access Databases, or MDB files), ODBC-supported data sources, and Indexed Sequential Access Method databases (ISAM, which includes Excel, dBase, and a few other databases).

Data Access Objects (DAO) was introduced as a means of interacting with Jet. DAO, through COM, provided an object-oriented interface to Jet and Microsoft Access.

Jet and DAO were successful in their flexibility but added layers to the ODBC API and were therefore more efficient for some databases (Access/MDB and ISAM) than others, including Relational Database Management Systems (RDBMS). DAO is still widely used today, but it is most appropriate for single-user, low-traffic database applications. The problem with DAO, as many soon began to see, was that it was so full-featured that it brought with it a profusion of objects. Figure 1-1 shows the DAO object model.

As you will see later in this chapter and in other chapters, ADO was designed to address this and other problems with DAO.
Microsoft’s response to the developer’s need for easier access to ODBC data sources came, in 1995, in the form of Remote Data Objects, or RDO. RDO provided more direct, and thus faster, access to the ODBC API, as well as support for RDBMS sources. With RDO, the emphasis moved from data-access methods designed for ISAM databases toward techniques to provide for stored
procedures and the results that they returned. RDO lacked some of the power that DAO offered with Jet (for instance, RDO is not designed to access ISAM sources and does not allow the creation of new databases), but it offered more power for newer, more robust enterprise systems.

The problem with RDO is that it is very different from the DAO architecture, which means two things. First, developers had to learn a new interface, and second, converting an existing DAO application to RDO involved a lot of additional development, because almost every piece of RDO differed from DAO, as you can see by comparing Figure 1-1 and Figure 1-2 (the RDO object model). With the introduction of RDO, developers chose between DAO and RDO instead of moving directly to RDO and abandoning DAO.

```
Figure 1-2. The RDO object model
```

**ODBCDirect**

ODBCDirect was provided as part of a later release of DAO; to save time, it allows developers to work directly with Access sources without using Jet as the go-between. It is similar to RDO’s direct access but includes RDO’s direct access to remote data sources.
OLE DB

ODBC provides access only to relational databases. Its successor, Object Linking and Embedding Databases (OLE DB), includes all other data sources. OLE DB is the foundation upon which ADO relies.

OLE DB provides the following features:

- Access to data regardless of its format or location (via COM—see “ADO and COM: Language Independence” later in this chapter)
- Full access to ODBC data sources and ODBC drivers
- A specification that Microsoft wants to act as a standard throughout the industry

OLE DB comprises four types of components; Figure 1-3 shows their relationships, which are described here:

Data consumer
Any application or tool that accesses data from a data source. While the API calls that are available to access the data in your database are considered data providers, the application that uses that data itself is a data consumer, since it requests the data from the data provider.

Data service provider
The engine that makes OLE DB work; the resource necessary for a data provider to be able to provide data. A data service provider is a modular or add-on component that allows an application to deliver data through OLE DB. Data service providers are usually provided by the vendor for major products such as Oracle, DB2, and Informix. Microsoft promotes the creation of data service providers by either the manufacturer of the data provider or a third-party company.

Business component
A go-between for a data provider and a data consumer. In today’s development environment, it is becoming more and more important not to develop in such a way that every object in your application manipulates your data. With a business component that you call to access your data, which in turn calls your database access component (ADO, RDO, ODBC, OLE DB, or ADO), then you need only modify the code in that business component.

Data provider
A component (application or database engine, for example) that delivers data from a data source (such as a database, spreadsheet, or email message) in a consistent manner.
ODBC, as we have just seen, is an excellent technology for accessing SQL-based data. OLE DB incorporates this proven technology with a particular component that allows OLE DB consumers to communicate directly with ODBC providers. In other words, use OLE DB to access SQL-based data, and you gain the advantage of being able to access both relational and other forms of data with the same code.

As they have done with ODBC, Microsoft is actively encouraging software vendors and tool developers to support the OLE DB standard within their applications and tools. Widespread standardization is an advantage for developers; with OLE DB, we can ensure that our applications become more robust and more powerful as they span the enterprise.

Keep in mind that OLE DB was designed for software vendors who develop data-based applications to expose that data to you, an end-user developer, through a consistent interface. OLE DB is fast, efficient, and powerful. It has everything a developer looks for in a data-access technology. It offers access to any data source known to man (or to Windows, for that matter), and it provides access to these data sources with a consistent interface, regardless of data source. The problem with OLE DB is that, like ODBC, it is inaccessible to Visual Basic and other developers, because it is based on a C-style API. Visual Basic developers, in particular, needed more.

**ADO**

Enter ActiveX Data Objects (ADO). ADO, a new application-level interface to OLE DB, is the latest, greatest piece of Microsoft’s UDA strategy. It combines the best features of its predecessors, DAO and RDO, and adds OLE DB accessibility for VBA.
ADO comprises a collection of object libraries in a new, modular object model: in this new model, many objects can exist independently of the others, as you will see in later chapters of this book. The ADO object model is more flexible than the DAO object model, but it’s similar, so programmers familiar with DAO will feel at home with ADO. ADO is a smaller version of DAO, generalized to allow easy access to any data source, not just Jet databases or ODBC data sources. The ADO object model simplifies data access more than DAO or RDO did by using fewer objects. See Figure 1-1 and also Chapter 2, *The ADO Architecture*, for more information.

Used with OLE DB, ADO provides fast, simple access to almost any data source. It allows developers to use a single, consistent interface to new and legacy databases and other data sources of all formats, when creating desktop—or web-based—applications.

ADO can also use the OLE DB provider for ODBC. Instead of removing the already proven and tested code for ODBC drivers, ADO allows you to use ODBC through the same interface you would for OLE DB. This may be an option when you have code you are migrating from RDO, which already uses ODBC.

ADO breaks the common characteristics of all data sources into easy-to-use components (which we will look at in Chapter 2). Consistency and language-independence are provided, so that developers can worry more about the content and quality of applications, rather than about the techniques used in delivering data or the type of data being used.

What does language-independent development mean? It is quite simple—one technology, one development interface. You will use the same object, method, and property names with ADO, regardless of the development language that you are using. The difference is almost unnoticeable. Under the covers, ADO, through COM (Component Object Model), worries about the particular language you are developing with, whether it is Visual Basic, Visual C++, or Java. Even scripting languages, such as VBScript and JavaScript in HTML pages are supported. We will look more closely into programming for these different languages in Chapter 3, *Accessing ADO with Various Languages*.

With this feature, you might expect that a lot of specific functionality of data sources would be lost. On the contrary, ADO allows the developer to access any data source-specific commands, methods, properties, and utilities that the vendor
has made available through OLE DB. And yes, ADO does this in a well-structured, consistent way. Can you possibly ask for more?

As we will see in chapters to come, an application can be designed to access a simple database, such as Access, and with a little bit of additional code, it can later access more intricate databases, such as SQL Server databases, Word documents, or email files. The only real coding necessary involves altering the connection string used in ADO to read the new data source. This powerful technology will help us move into the future as applications begin to grow across enterprises.

**RDS**

The final piece of data-access technology in this list of the MDAC components is Remote Data Services (RDS). RDS, based on existing Active Data Connector (ADC) technology integrated into ADO, transports ADO objects via proxy between server and client, thus allowing developers to create web-based applications that can access data on the server in new ways. Some of the advantages of RDS are:

- Client-side caching of data results
- Ability to update data from the client
- Support for data-aware ActiveX components and controls

Client-side caching is something that we will all grow to love. With it, clients (end-users) are able to view data from the server without making numerous round trips. For instance, when you are using a search engine on the Internet, such as Yahoo!, you receive a list of links that relate to your search, usually in groups of tens. If you want to see the next ten sites from the resulting search, your browser must make another request to the server. With client-side caching, all of the data is sent to the client, so that the client can browse this data without incurring time delays that are associated with additional requests. This feature reduces local-area network and Internet traffic and allows the end-user to move freely through data without unnecessary pauses and to perform operations on that data, such as sorting and filtering.

With RDS, web pages can now offer the client the ability to interact with and alter data. This data can be sent back to the server after manipulation. At the server, the data can be verified and then returned to the data source. With this technology, your client/server applications can span the Internet (or your intranet). Clients can now invoke server-side automation objects through HTML, meaning that particular business rules (chosen by the developer) can be accessed via the client.

RDS enables three-tier client/server applications, with the model shown in Figure 1-4.
With automation objects, your application can become an auto-downloaded application. For businesses with a large number of client-side users, you can create, maintain, and update your application on the server alone. When clients run your application, they can use an ActiveX-aware browser (Internet Explorer) to access the application. With auto-download features built into the browser, the client receives an updated version of the application.

RDS also supports data-aware ActiveX controls that can be placed within an HTML page on a client. For instance, if you want to allow the client to view a list of documents that you have stored in your data source on the server, you could link RDS to an ActiveX list box control that is placed in the HTML page and downloaded to the client. The control interacts automatically with RDS, without any additional programming, to download all of the document names.

See Chapter 11, Remote Data Services, for a more detailed introduction to RDS.

**Putting It All Together**

With the addition of RDS to the VISTA family of components, Microsoft has integrated several useful existing technologies into the universal data-access strategy: IE data-access technology for data-bound web pages, remote data capability...
through RDS, and ASP/IIS-related technologies for better access to data services via the Internet. The result allows applications to work with data offline to reduce network traffic, update data on remote clients, and gather data asynchronously for faster response time.

Figure 1-5 shows the relationships and dependencies of the MDAC components.

As you can see from Figure 1-5, your application can use a number of different Microsoft-supplied technologies to access SQL—as well as non-SQL and legacy—data, such as that residing on a mainframe.

Until ADO, we had four choices: DAO, RDO, ODBC, and OLE DB. DAO served its purpose well: it used the power of the underlying (Jet) database engine to access Microsoft and other ISAM data sources. With RDO, things were even better with its easy-to-use interface to ODBC and ability to access almost any SQL data source. Accessing ODBC directly was always a possibility, but it was questionable whether the overwhelming amount of work was worth the extra speed gained in the process. Finally, OLE DB offered access to ISAM, SQL, non-SQL, and legacy data. However wonderful OLE DB was, it is considered the most difficult interface with which to develop to access data sources. This is
where ADO comes into play. ADO reports directly to OLE DB and no one else, meaning that it provides an interface to the whole complicated mess, about which we need to know little or nothing.

ADO provides a consistent development interface to the wonders of OLE DB, and it does so while being language-independent.

**ADO and COM: Language Independence**

Microsoft’s Component Object Model, better known as COM, is a mature technology that offers universal access to components, regardless of the language in which they were programmed. This is the backbone that allows ADO, through OLE DB, to be so versatile. To understand how COM allows ADO to be language-independent, you must first understand what COM is and what it achieves.

**COM**

COM is technology specification for writing software components that interact through a standard interface. The COM specification is strictly a binary specification. This guarantees that the language in which a COM object is developed has absolutely no importance once the object is compiled, as long as it adheres to the binary specification.

The COM specification sets rules for creating and managing component objects. This specification guarantees that all COM objects are compatible and that they expose a minimal set of interfaces. These interfaces allow COM objects to communicate with each other whether they are on the same machine or supported by networks. Since the COM specification relies on binary compatibility, COM works across heterogeneous networks. In other words, COM objects can run on any machine, even without the Windows operating system.

A particular type of COM implementation is **OLE Automation**, or simply Automation. Automation is a standard way for COM objects to expose their functionality to software products, development languages, and even scripting languages. The use of Automation allows applications to actually manipulate other applications through the exposed features and functionality of the latter’s COM objects. Automation allows two applications to communicate with each other.

An example of this type of manipulation is a Visual Basic add-in. Visual Basic exposes an object model through the COM technology to any other component that wishes to interact with it. You can create an add-in for Visual Basic that works seamlessly with the product through use of Visual Basic’s exposed features. As a matter of fact, many of Microsoft’s products expose their features through COM, including the Microsoft Office family of products. Microsoft Word, for example,
exposes its functionality through COM and allows itself to be manipulated through scripting with VBA (Visual Basic for Applications).

When a COM object is exposed through OLE Automation, that object is then called an **ActiveX object** or an **ActiveX server**. The application or tool that manipulates the ActiveX object is called an **ActiveX client**.

### ADO and COM

As a COM technology, ADO has the ability to communicate with any data source that provides an OLE DB interface. ADO and OLE DB share the same backbone—COM. Figure 1-6 shows COM at work with ADO and OLE DB. When ADO communicates with a data provider at the simplest level, two COM objects are exchanging information, regardless of the connection between them.

![Figure 1-6. ADO and COM](image)

Also, COM has the ability to send events or notifications to other COM objects. This capability is used in ADO, as we will see later on when we execute queries. We have the ability, through ADO, OLE DB, and finally COM, to send a request for a selection of records through SQL and then to be notified when it has completed processing.

What is even better is that COM has been around for a long time, has gained the respect of application and tools developers, has a proven track record, and is supported by Microsoft. ADO's architecture does not change between programming languages or operating systems, just as does COM.
COM objects are easily distributed. They have the ability to communicate across machines and enterprises. This advantage is embraced with ADO through RDS, or Remote Data Service, which I will be talking about in Chapter 11.

As you can see from this very limited introduction to COM, ADO stands upon OLE DB, which relies heavily on COM to communicate with other COM objects. This can do nothing but benefit us as developers, because it enables communication with objects that aren’t necessarily written in the same language.*

When to Use ADO

ADO is language-independent, as discussed earlier. This means that no matter which language you are developing with—Visual Basic, VBScript, Visual Basic for Applications (VBA), Visual C++, Visual J++, or JavaScript—the development interface is identical. This allows developers to become familiar with the technology itself, instead of worrying about learning a half-dozen different programming syntaxes for that technology. I suggest that you use ADO whenever your application fits into any or all of the following categories:

- Your application accesses or may later need to access more than one data source.
- Your application accesses or may later need to access data sources other than ISAM or ODBC databases.
- Your application spans or may later span a heterogeneous network.
- Your application uses or may later use multiple languages.

If your application needs to access more than one type of data source, then you should consider integrating ADO technology into your application. For instance, if you were designing an application that had to search Word documents, email messages, and a SQL Server database for keywords and then to show related information based on that query, ADO is the best choice. With ADO, you can create a component to search all three of these data sources using identical code, saving you time in development, as well as in maintenance and upkeep. This choice also provides the option of adding a fourth data source to your application at some later time with little or no additional overhead in development.

If your application may access data sources other than conventional ISAM or ODBC databases, you should use ADO. With ADO, you can search through an Excel worksheet just as if you were searching through email messages. If you use some other technology besides ADO, you must not only code two different

---

* For more information, see Inside COM by Dale Rogerson (Microsoft Press, 1997).
About the Author

Jason T. Roff is the author of three database-development books, including *ADO: ActiveX Data Objects*. He specializes in Visual Basic, ASP, and SQL Server development and architecture, and has experience working with C++ and Assembly on everything ranging from a Commodore to a Unix box. Jason graduated from the University of Albany with a degree in computer science with applied mathematics. Currently, he manages local and offsite development teams to create web- and Windows-based applications.

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The bird on the cover of *ADO: ActiveX Data Objects* is an ivory-billed woodpecker (*Campephilus principalis*). Considered extinct by many naturalists and ornithologists (the last confirmed sighting was in the 1950s), the “ivory-bill” was never abundant in its habitat, the southeastern United States and Cuba. With glossy black plumage, white markings, and a red tufted crest (males only), the ivory-bill looks extremely similar to the pileated woodpecker, with whom it also shared its habitat. The similarities between the two birds have been the cause of much trouble, as eager amateurs add to unconfirmed sighting reports of the ivory-bill when they have probably spotted the pileated woodpecker. This is especially troublesome for naturalists who hold out hope that the ivory-bill may still exist in the far reaches of Louisiana forests or in Cuba. In the early 1990s, many nature and birding groups spent considerable amounts of money mounting search efforts for the ivory-bill.

As do all woodpeckers, the ivory-bill has a chisel-like bill and a long, hard-tipped, sticky tongue; the first for drilling and scaling bark, the latter for retrieving beetles and grubs on which to feed. Retrieving food in this manner, however, is not what creates the drumming sound that many associate with woodpeckers. Rather, woodpeckers drum when reinforcing their claim to a territory, creating the loudest drum possible by striking the tops of dead, hollow trees.

Important differences between the closely linked ivory-billed and pileated woodpeckers include their bills (the ivory-bill’s was, well, ivory, while the pileated woodpecker’s bill is gray), size (the ivory-bill was the largest of all North
American woodpeckers), and their calls (the ivory-bill’s was a “toot”; the pileated’s is a “kuk”). In 1987, Dr. Jerome A. Jackson of Florida Gulf University caught the ivory-bill’s distinctive call on eighteen minutes of tape in Louisiana, adding to the excitement created by various unconfirmed sightings. The most recent and credulous sighting occurred in 1999, when graduate student David Kulivan sighted a pair of what were supposedly ivory-bills in southeastern Louisiana.

While The Nature Conservancy declared the ivory-bill extinct in 1994, the U.S. Fish and Wildlife Service has not yet added it to its extinction list. The reason for its near or possible extinction: logging of the old-growth forests in which it lived.

Jeffrey Holcomb and Sarah Jane Shangraw were the production editors for *ADO: ActiveX Data Objects*. Jeffrey Holcomb copyedited the text. Linley Dolby, Matt Hutchinson, and Claire Cloutier provided quality control. Pamela Murray, Sarah Jane Shangraw, and Joe Wizda wrote the index. Sarah Jane Shangraw did page composition.

Hanna Dyer designed the cover of this book, based on a series design by Edie Freedman. The cover image is a 19th-century engraving from the Dover Pictorial Archive. Erica Corwell produced the cover layout with QuarkXPress 4.1 using Adobe’s ITC Garamond font.

Melanie Wang designed the interior layout based on a series design by Nancy Priest. Anne-Marie Vaduva converted the files from Microsoft Word to FrameMaker 5.5.6 using tools created by Mike Sierra. The text and heading fonts are ITC Garamond Light and Garamond Book; the code font is Constant Willison. The illustrations that appear in the book were produced by Robert Romano and Jessamyn Read using Macromedia FreeHand 9 and Adobe Photoshop 6. This colophon was written by Jeffrey Holcomb.