DBMaker

OLEDB User's Guide

CASEMaker®

CASEMaker Inc./Corporate Headquarters

1680 Civic Center Drive

Santa Clara, CA 95050, U.S.A.

www.casemaker.com

www.casemaker.com/support

Document No. 43/DBM43-T01232006-01-OLED

Publication Date: 2006-01-23

For a description of updated functions that do not appear in this manual, read the file named README.TXT after installing the CASEMaker DBMaker software.

Trademarks

CASEMaker, the CASEMaker logo, and DBMaker are registered trademarks of CASEMaker Inc. Microsoft, MS-DOS, Windows, and Windows NT are registered trademarks of Microsoft Corp. UNIX is a registered trademark of The Open Group. ANSI is a registered trademark of American National Standards Institute, Inc.

Other product names mentioned herein may be trademarks of their respective holders and are mentioned only form information purposes. SQL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

Notices

The software described in this manual is covered by the license agreement supplied with the software.

Contact your dealer for warranty details. Your dealer makes no representations or warranties with respect to the merchantability or fitness of this computer product for any particular purpose. Your dealer is not responsible for any damage caused to this computer product by external forces including sudden shock, excess heat, cold, or humidity, nor for any loss or damage caused by incorrect voltage or incompatible hardware and/or software.

Information in this manual has been carefully checked for reliability; however, no responsibility is assumed for inaccuracies. This manual is subject to change without notice.

Content

1 Introduction

OLE DB is a set of Component Object Model (COM) interface that provide applications with uniform access to data stored in diverse DBMS and non-DBMS information sources and that also provide the ability to implement additional database services. Utilizing these interfaces the Data Consumer can access data through a same method, without considering the place that data are stored, the format of data, and the type of data.

OLE DB Provider for DBMaker is designed for accessing the DBMaker database system. OLE DB programmer can develop consumer application with high performance through the interfaces provided by OLE DB Provider for DBMaker. OLE DB Provider for DBMaker is specifically for DBMaker, and consumer program should not use this provider to access any other information source.

2 COM Objects Defined in OLE DB Provider for DBMaker

OLE DB uses the COM infrastructure. It is the Microsoft standard for Universal Data Access. Like the ODBC system, OLE DB also provides a set of APIs. But OLE DB APIs are completely based on COM, and the communication based on COM can be reckoned operations on some abstract objects, such as Data source, Session, Command, and Rowset. OLE DB Provider for DBMaker supports four objects; they are Data source Object, Session Object, Command Object, and Rowset Object. These objects are described in the following section of this chapter.

2.1 Data Source

In OLE DB, a data source object is a COM object through which a consumer connects to a provider's underlying data store. OLE DB Provider for DBMaker defines its own data source object class. To connect to the provider, a consumer must create and initialize an instance of this class. Data source objects are factories for session objects.

The data source object cotype is defined as follows. For more information about the interfaces, see "Interfaces implemented in Data source" in Chapter 3, "Interfaces supported by OLE DB Provider for DBMaker".

```
CoType TDataSource {
        [mandatory]     interface IDBCreateSession;
        [mandatory]     interface IDBInitialize;
        [mandatory]     interface IDBProperties;
        [mandatory]     interface IPersist;
        [optional]      interface IConnectionPointContainer;
        [optional]      interface IDBInfo;
        [optional]      interface IPersistFile;
    }
```

2.2 Sessions

Session object represents a single connection to a DBMaker database. The session object exposes the interfaces that allow data access and manipulation. A single data source object may be able to create multiple sessions. Session objects are a factory for command and rowset objects, which provide methods for creating command objects and rowsets and modifying tables and indexes. Session objects can also act as factories for transaction objects, which are used to control nested transactions.

After all references to the session object are released, the session object is removed from memory and the connection is dropped.

The session object cotype is defined as follows. For more information about the interfaces, see “Interfaces implemented in Session” in Chapter 3, “Interfaces supported by OLE DB Provider for DBMaker”.

```
CoType TSession {
        [mandatory]     interface IGetDataSource;
        [mandatory]     interface IOpenRowset;
        [mandatory]     interface ISessionProperties;
        [optional]      interface IAlterIndex;
        [optional]      interface IAlterTable;
        [optional]      interface IBindResource;
        [optional]      interface ICreateRow;
        [optional]      interface IDBCreateCommand;
        [optional]      interface IDBSchemaRowset;
        [optional]      interface IIndexDefinition;
        [optional]      interface ISupportErrorInfo;
        [optional]      interface ITableCreation;
        [optional]      interface ITableDefinition;
        [optional]      interface ITableDefinitionWithConstraints;
        [optional]      interface ITransaction;
        [optional]      interface ITransactionJoin;
        [optional]      interface ITransactionLocal;
        [optional]      interface ITransactionObject;
    }
```

2.3 Command

Commands can be in one of four states: Initial, Unprepared, Prepared, or Executed. Parameters can be used with commands to bind to consumer variables at execution time. When executed, a command returns either a single result, which can be either a rowset object or a row count, which is the number of rows affected by a command that updates, deletes, or inserts rows. A command also

can return multiple results, which must be returned in a multiple results object, if the command text comprises multiple, separate text commands—such as a batch of SQL statements—or if more than one set of parameters is passed to a command.

The command object is used to execute a provider for DBMaker text command. Text command are expressed in the provider for DBMaker language, and are generally used for creating a rowset—for example, executing an SQL SELECT statement.

The command object cotype is defined as follows. For more information about the interfaces, see "Interfaces implemented in Command" in Chapter 3, "Interfaces supported by OLE DB Provider for DBMaker".

```
CoType TCommand {
        [mandatory]     interface IAccessor;
        [mandatory]     interface IColumnsInfo;
        [mandatory]     interface ICommand;
        [mandatory]     interface ICommandProperties;
        [mandatory]     interface ICommandText;
        [mandatory]     interface IConvertType;
        [optional]      interface IColumnsRowset;
        [optional]      interface ICommandPersist;
        [optional]      interface ICommandPrepare;
        [optional]      interface ICommandWithParameters;
        [optional]      interface ISupportErrorInfo;
    }
```

2.4 Rowset

Rowsets are the central objects that enable OLE DB components to expose and manipulate data in tabular form. A rowset object is a set of rows in which each row has columns of data. For example, the provider for DBMaker presents data, as well as metadata, to consumers in the form of rowsets. Query processor presents query results in the form of rowsets. The use of rowsets throughout OLE DB makes it possible to aggregate components that consumer or produce data through the same object.

The rowset object cotype is defined as follows. For more information about the interfaces, see “Interfaces implemented in Rowset” in Chapter 3, “Interfaces supported by OLE DB Provider for DBMaker”.

```
CoType TRowset {
        [mandatory]     interface IAccessor;
        [mandatory]     interface IColumnsInfo;
        [mandatory]     interface IConvertType;
        [mandatory]     interface IRowset;
        [mandatory]     interface IRowsetInfo;
        [optional]      interface IConnectionPointContainer;
        [optional]      interface IDBAsynchStatus;
        [optional]      interface IRowsetChange;
        [optional]      interface IRowsetFind;
        [optional]      interface IRowsetIndex;
        [optional]      interface IRowsetLocate;
        [optional]      interface IRowsetRefresh;
        [optional]      interface IRowsetScroll;
    [optional]      interface IRowsetUpdate;
    [optional]      interface IRowsetView;
      }
```

3 Interfaces Supported by OLE DB Provider for DBMaker

Interfaces are a group of semantically related functions that provide access to a COM object. Each OLE DB interface defines a contract that allows objects to interact according to the Component Object Model (COM). While OLE DB provides many interface implementations. Most interfaces can also be implemented by developers designing OLE DB applications.

This chapter contains the information for all OLE DB interfaces and methods supported by OLE DB provider for DBMaker.

3.1 Interfaces Implemented in Data Source

This section introduces the interfaces implemented in data source object.

3.1.1 IDBCreateSession

Consumers call IDBCreateSession::CreateSession on a data source object to obtain a new session.

Method	Description
CreateSession	Creates a new session from the data source object and returns the requested interface on the newly created session.

3.1.2 IDBInitialize

IDBInitialize is used to initialize and uninitialize data source objects. It is a mandatory interface on data source objects.

Method	Description
Initialize	Initializes a data source object
Uninitialize	Returns the data source object to uninitialized state.

3.1.3 IDBProperties

IDBProperties is used to set and get the values of properties on the data source object and to get information about all properties supported by the provider of DBMaker.

A data source object that has not been initialized or that has been uninitialized is in an uninitialized state. When a data source object is in this state, the consumer can work only with properties of that object that belong to the Initialization property group. After the data source object is initialized, it is in an initialized state (until it is uninitialized). When the data source object is in this state, the consumer can work with properties of that object that belong to the Initialization, Data Source, and Data Source Information property groups. However, the consumer cannot set the values of properties in the Initialization property group.

Only properties in the Initialization property group are guaranteed to survive uninitialization. That is, if the consumer uninitializes and reinitializes a data source object, it might need to reset the values of properties in groups other than Initialization.

IDBProperties is a mandatory interface for data source objects.

Method	Description
GetProperties	Returns the values of properties in the Data Source, Data Source Information, and Initialization property groups that are currently set on the data source object.

Method	Description
GetPropertyInfo	Returns information about all properties supported by the provider for DBMaker.
SetProperties	Sets properties in the Data Source and Initialization property groups, for data source object.

3.1.4 IPersist

The IPersist interface defines the single method GetClassID, which is designed to supply the CLSID of an object that can be stored persistently in the system. The CLSID is a unique value that identifies the code that can manipulate the persistent data.

The single method of IPersist is rarely called directly in application code. It is called by the default object handler to get the CLSID of an embedded object, or an object to be marshaled.

Method	Description
GetClassID	Returns the class identifier (CLSID) for the component object.

3.1.5 IConnectionPointContainer

The IConnectionPointContainer interface supports connection points for connectable objects. When implemented on an object, makes the object connectable and expresses the existence of outgoing interfaces on the object. Through this interface a client can locate a specific connection point for one IID. (This is an OLE, not an OLE DB, interface.)

The consumers call IConnectionPointContainer::EnumConnectionPoints to create an enumerator object to iterate through all the connection points supported in the connectable object, one connection point per outgoing IID.

The connection point is a separate sub-object to avoid circular reference counting problems.

Method	Description
EnumConnectionPoints	Returns an object to enumerate all the connection points supported in the connectable object.
FindConnectionPoint	Returns a connection pointer to a specific IID.

3.1.6 IDBInfo

IDBInfo returns information about the keywords and literals the provider for DBMaker supports. It is an optional interface on the data source objects.

Method	Description
GetKeywords	Returns a list of provider-specific keywords.
GetLiteralInfo	Returns information about literals used in text commands and the definition of the tables and the indexes.

3.1.7 IPersistFile

The **IPersistFile** interface provides methods that permit an object to be loaded from or saved to a disk file, rather than a storage object or stream. Because the information needed to open a file varies greatly from one application to another, the implementation of **IPersistFile::Load** on the object must also open its disk file.

The IPersistFile interface inherits its definition from IPersist, so all implementations must also include the GetClassID method of IPersist.

Implement IPersistFile when you want to read or write information from a separate file, which could be of any file format.

When IPersistFile is implemented on an object that supports linking through a file moniker and the application for the linked object is run, OLE calls IPersistFile::Load. Once the file is loaded, OLE calls IPersistFile::QueryInterface to get another interface pointer to the loaded object. The IPersistFile interface is typically part of an aggregate object that offers other interfaces.

Method	Description
IsDirty	Checks an object for changes since it was last saved to its current file.
Load	Opens the specified file and initializes an object from the file contents
Save	Save the object into the specified file.
SaveCompleted	Notifies the object that it can revert from NoScribble mode to Normal mode.
GetCurFile	Gets the current name of the file associated with the object.

3.2 Interfaces Implemented in Session

This section introduces the interfaces implemented in data source object.

3.2.1 IGetDataSource

This is a mandatory interface on the session for obtaining an interface pointer to the data source object.

Method	Description
GetDataSource	Returns an interface pointer on the data source object that created this session.

3.2.2 IOpenRowset

IOpenRowset is a required interface on the session. It can be supported by providers that do not support creating rowsets through commands. Consumer can create a Rowset Object directly using this interface.

The IOpenRowset model enables consumers to open and work directly with individual tables or indexes in a data store by using IOpenRowset::OpenRowset, which generates a rowset of all rows in the table or indexes.

Method	Description
OpenRowset	Opens and returns a rowset that includes all rows from a single base table or index.

3.2.3 ISessionProperties

ISessionProperties returns information about the properties a session supports and the current settings of those properties. It is a mandatory interface on sessions.

Method	Description
GetProperties	Returns the list of properties in the Session property group that are currently set on the session.
SetProperties	Sets properties in the Session property group.

3.2.4 IAlterIndex

The **IAlterIndex** interface exposes methods to alter tables, indexes, and columns.

Method	Description
AlterIndex	Alters the index ID and/or properties associated with an index.

3.2.5 IAlterTable

The **IAlterTable** interface exposes methods to alter the definition of tables, indexes, and columns.

When implement this interface, the provider should expose the data source information property DBPROP_ALTERCOLUMN.

Method	Description
AlterColumn	Alters the column ID and/or properties associated with a column in a table.
AlterTable	Alters the table ID and/or properties associated with a table.

3.2.6 IBindResource

IBindResource binds to an object named by a URL and returns a data source, session, rowset, row, command, or stream object.
NOTE: The term "resource" in the name of this interface refers to the use of URLs to name OLE DB objects.

IBindResource is a mandatory interface session objects.

Method	Description
Bind	Binds to an object named by a URL.

3.2.7 ICreateRow

ICreateRow creates and binds to an object named by a URL. It is an optional interface on session and row objects.

Consumers call **ICreateRow::CreateRow** to create and bind to an object named by a URL.

Method	Description
CreateRow	Creates and binds to an object named by a URL and returns the requested interface pointer.

3.2.8 IDBCreateCommand

Consumers call **IDBCreateCommand::CreateCommand** on a session to obtain a new command.

Method	Description
CreateCommand	Creates a new command.

3.2.9 IDBSchemaRowset

This is an optional interface on session. It is used to provide advanced schema information.

Consumers can get information about a data store without knowing its structure by using the **IDBSchemaRowset** methods.

IDBSchemaRowset::GetRowset allows consumers to specify simple restrictions, such as returning all the columns in a particular table.

The provider must return all of the columns in the rowsets they return. If they cannot return the information in a column, they must return an appropriate value. Generally, this is NULL. The provider can return provider-specific columns after the last column defined by OLE DB.

Method	Description
GetRowset	Returns a schema rowset.
GetSchemas	Returns a list of schema rowsets accessible by **IDBSchemaRowset::GetRowset**.

3.2.10 IIndexDefinition

IIndexDefinition exposes simple methods to create and drop indexes from the data store.

Method	Description
CreateIndex	Adds a new index to a base table.
DropIndex	Drops an index from a base table.

3.2.11 ISupportErrorInfo

ISupportErrorInfo is defined by Automation. This interface indicates whether a specific interface can return Automation error objects. Because OLE DB error objects are returned through the same mechanism as Automation error objects, support for them is also indicated through this interface.

ISupportErrorInfo must be implemented by the provider for DBMaker on any object that exposes an interface that can return OLE DB error objects. A consumer uses this interface to determine whether a particular OLE DB interface can return an OLE DB error object.

Method	Description
InterfaceSupportsErrorInfo	Indicates whether a specific OLE DB interface can return OLE DB error objects.

3.2.12 ITableCreation

The **ITableCreation** extends ITableDefinition to provide a full set of method to create or describe the complete set of table definition information.

Method	Description
GetTableDefinition	Adds a new column to a base table.

3.2.13 ITableDefinition

The **ITableDefinition** interface exposes simple methods to create, drop, and alter tables on the data store.

Method	Description
AddColumn	Adds a new column to a base table.
CreateTable	Creates a new base table in the data store.
DropColumn	Drops a column from a base table.
DropTable	Drops a base table in the data store.

3.2.14 ITableDefinitionWithConstraints

The **ITableDefinitionWithConstraints** interface exposes simple methods to create, drop and alter tables on the data store. It also provides a mechanism to create and delete multitable and multicolumn constraints on tables.

ITableDefinitionWithConstraints extends **ITableCreation** by adding the ability to creation and drop constraints from the base table and create base tables with constraints in one, atomic operation.

Method	Description
CreateTableWithConstraints	Creates a new base table in the data store, allowing constraints to be added atomically during table creation.
AddConstrain	Adds a new constraint to a base table.
DropConstraint	Drops a constraint from a base table.

3.2.15 ITransaction

The **ITransaction** interface is used to commit, abort, and obtain status information about transactions.

Method	Description
Abort	Aborts a transaction.
Commit	Commits a transaction.
GetTransactionInfo	Returns information about a transaction.

3.2.16 ITransactionJoin

The provider for DBMaker supports distributed transactions. So the **ITransactionJoin** interface can be exposed by the provider.

Method	Description

Method	Description
GetOptionObject	Returns an object that can be used to specify configuration options for a subsequent call to **JoinTransaction**.
JoinTransaction	Requests that the session enlist in a coordinated transaction.

3.2.17 ITransactionLocal

ITransactionLocal is an optional interface on sessions. It is used to start, commit, and abort transactions on the session.

Method	Description
GetOptionObject	Returns an object that can be used to specify configuration options for a subsequent call to **StartTransaction**.
StartTransaction	Begins a new transaction.

3.2.18 ITransactionObject

ITransactionObject enables consumers to obtain the transaction object associated with a particular transaction level.

Method	Description
GetTransactionObject	Returns an interface pointer on the transaction object.

3.3 Interfaces Implemented in Command

This section introduces the interfaces implemented in data source object.

3.3.1 IAccessor

IAccessor provides methods for accessor management. All rowsets and commands must implement **IAccessor**.

An accessor is a data structure created by the consumer that describes how row or parameter data from the data store is to be laid out in the consumer's data buffer. For each column in a row (or parameter in a set of parameters), the accessor contains a binding. A binding is a data structure that holds information about a column or parameter value, such as its ordinal value, data type, and destination in the consumer's buffer.
NOTE: To create an accessor, a consumer calls **IAccessor::CreateAccessor**. The consumer may create and release accessor at any time while the rowset or command remains in existence. When one thread of a consumer shares an accessor with another thread, it calls **IAccessor::AddRefAccessor** to increment the reference count of that accessor.
NOTE: When the consumer is done with a rowset, it calls **IAccessor::ReleaseAccessor** to release any accessors on the rowset, including accessors inherited from the command. When the consumer is done with a command, it calls **IAccessor::ReleaseAccessor** to release any accessors created on the command. In either case, the consumer must call IAccessor::ReleaseAccessor once for each reference count on the accessors.

Method	Description
AddRefAccessor	Adds a reference count to an existing accessor.
CreateAccessor	Creates an accessor from a set of bindings.
GetBindings	Returns the bindings in an accessor.
ReleaseAccessor	Releases an accessor.

3.3.2 IColumnsInfo

NOTE: IColumnsInfo is required both on command and on rowsets. It is the simpler of two interfaces that can be used to expose information about columns of a rowset of prepared command. It provides a limited set of information in an array.
NOTE: IColumnsInfo::GetColumnInfo returns the most commonly used metadata: column IDs, data types, updatability, and so on.
NOTE: IColumnsInfo::GetColumnInfo returns the metadata in an array of structures, which can be created and accessed quickly. The metadata returned, however, is limited.

Method	Description
GetColumnInfo	Returns the column metadata needed by most consumers.
MapColumnIDs	Returns an array of ordinals of the columns in a rowset that are identified by the specified column IDs.

3.3.3 ICommand

ICommand contains methods to execute commands. A command can be executed many times, and the parameter values can vary. This interface is mandatory on commands.

A command object contains a single text command, which is specified through **ICommandText**.

Method	Description
Cancel	Cancels the current command execution.
Execute	Executes the command.
GetDBSession	Returns an interface pointer to the session that created the command.

3.3.4 ICommandProperties

ICommandProperties specifies to the command the properties from the Rowset property group that must be supported by the rowset returned by **ICommand::Execute**. A special case of these properties, and the ones most commonly requested, are the interfaces the rowset must support. In addition to interfaces, the consumer can request properties that modify the behavior of the rowset or interfaces.

All rowsets must support **IRowset**, **IAccessor**, **IColumnsInfo**, **IRowsetInfo**, and **IConvertType**. Provider may choose to return rowsets supporting other interfaces if doing so is possible and if the support for the returned interfaces does not affect consumer code that is not expecting them. The riid parameter of **ICommand::Execute** should be one of the interfaces returned by **IRowsetInfo::GetProperties**.

This interface is mandatory on commands.

Method	Description
GetProperties	Returns the list of properties in the Rowset property group that are currently requested for the rowset.
SetProperties	Sets properties in the Rowset property group.

3.3.5 ICommandText

This interface is mandatory on commands.

A command object can have only one text command. When the command text is specified through **ICommandText::SetCommandText**, it replaces the existing command text.

Method	Description
GetCommandText	Returns the text command set by the last call to ICommandText::SetCommandText.
SetCommandText	Sets the command text, replacing the existing command text.

3.3.6 IConvertType

This interface is mandatory on commands, rowsets, and index rowsets.

IConvertType contains a single method that gives information about the availability of type conversions on a command or on a rowset.

Method	Description
CanConvert	Gives information about the availability of the type conversions on a command or on a rowset.

3.3.7 IColumnsRowset

IColumnsRowset is optional both on commands and on rowsets. This interface supplies complete information about columns in a rowset. The methods in it can be called from a rowset or a command.

Consumers use the methods in **IColumnsRowset** for detailed and flexible information about the columns of a rowset.

Method	Description
GetAvailableColumns	Returns a list of optional metadata columns that can be supplied in a column's rowset.
GetColumnsRowset	Returns a rowset containing metadata about each column in the current rowset. The rowset is known as the *column metadata rowset* and is read-only.

3.3.8 ICommandPersist

Command objects support **ICommandPersist** for persisting the state of a command object. Persisting a command object does not persist any active rowsets that may have resulted from the execution of the command object, nor does it persist accessors, property settings, or parameter information associated with the command object.

Persisted commands can be enumerated through the PROCEDURES rowset. Persisted commands that can act as the source of a new command (that is, a table in an **SQL FROM** clause) can be enumerated through the VIEWS rowset.

Method	Description
DeleteCommand	Deletes a persisted command.
GetCurrentCommand	Returns the DBID of the current command.
LoadCommand	Loads a persisted command.
SaveCommand	Saves the current command.

3.3.9 ICommandPrepare

This optional interface encapsulates command optimization, a separation of compile time and run time, as found in traditional relational database systems. The result of this optimization is a command execution plan.

If the provider supports command preparation, by supporting this interface, commands must be in a prepared state prior to calling the following methods: **IColumnsInfo::GetColumnInfo, IColumnsInfo::MapColumnIDs**, **IColumnsRowset::GetAvailableColumns**, and **IColumnsRowset::GetColumnsRowset**.

Method	Description
Prepare	Validates and optimizes the current command.
Unprepare	Discards the current command execution plan.

3.3.10 ICommandWithParameters

The OLE DB provider for DBMaker supports parameters, so it must support **ICommandWithParameters**. Any provider that returns DBPROPVAL_SQL_ANSI92_INTERMEDIATE or DBPROPVAL_SQL_ANSI92_FULL for the DBPROP_SQLSUPPORT property can support parameters.

This optional interface encapsulates parameters. Parameters are scalar values, or a vector of scalar values, typically expressed in predicates but possibly supported by many providers in any scalar expression.

For scalar parameters of prepared commands, there is a presumption that different parameter values do not require different plans. In other words, a single preparation and its resulting plan are satisfactory for all possible values of scalar parameters.

Parameter values are set when a command is executed. Methods are included here to offer a means for setting and obtaining a list of parameters and their types.

Method	Description
GetParameterInfo	Gets a list of the command's parameters, their names, and their types.
MapParameterNames	Returns an array of parameter ordinals when given named parameters.
SetParameterInfo	Specifies the native data type of each parameter.

3.3.11 ISupportErrorInfo

ISupportErrorInfo is defined by Automation. This interface indicates whether a specific interface can return Automation error object. Because OLE DB error objects are returned through the same mechanism as Automation error objects, support for them is also indicated through this interface.

ISupportErrorInfo must be implemented by the provider for DBMaker on any object that exposes an interface that can return OLE DB error objects. A consumer uses this interface to determine whether a particular OLE DB interface can return an OLE DB error object.

Method	Description
InterfaceSupportsErrorInfo	Indicates whether a specific OLE DB interface can return OLE DB error objects.

3.3.12 IRowsetView

This section introduces the interfaces implemented in rowset object.

3.3.13 IAccessor

IAccessor provides methods for accessor management. All rowsets and commands must implement IAccessor.
An accessor is a data structure created by the consumer that describes how row or parameter data from the data store is to be laid out in the consumer's data buffer. For each column in a row (or parameter in a set of parameters), the accessor contains a binding. A binding is a data structure that holds information about a column or parameter value, such as its ordinal value, data type, and destination in the consumer's buffer.

To create an accessor, a consumer calls **IAccessor::CreateAccessor**. The consumer may create and release accessor at any time while the rowset or command remains in existence. When one thread of a consumer shares an accessor with another thread, it calls **IAccessor::AddRefAccessor** to increment the reference count of that accessor.

When the consumer is done with a rowset, it calls **IAccessor::ReleaseAccessor** to release any accessors on the rowset, including accessors inherited from the command. When the consumer is done with a command, it calls **IAccessor::ReleaseAccessor** to release any accessors created on the command. In either case, the consumer must call **IAccessor::ReleaseAccessor** once for each reference count on the accessors.

Method	Description
AddRefAccessor	Adds a reference count to an existing accessor.
CreateAccessor	Creates an accessor from a set of bindings.
GetBindings	Returns the bindings in an accessor.
ReleaseAccessor	Releases an accessor.

3.3.14 IColumnsInfo

IColumnsInfo is required both on command and on rowsets. It is the simpler of two interfaces that can be used to expose information about columns of a rowset of prepared command. It provides a limited set of information in an array.

IColumnsInfo::GetColumnInfo returns the most commonly used metadata: column IDs, data types, updatability, and so on.
IColumnsInfo::GetColumnInfo returns the metadata in an array of structures, which can be created and accessed quickly. The metadata returned, however, is limited.

Method	Description
GetColumnInfo	Returns the column metadata needed by most consumers.
MapColumnIDs	Returns an array of ordinals of the columns in a rowset that are identified by the specified column IDs.

3.3.15 IConvertType

This interface is mandatory on commands, rowsets, and index rowsets.

IConvertType contains a single method that gives information about the availability of type conversions on a command or on a rowset.

Method	Description
CanConvert	Gives information about the availability of type conversions on a command or on a rowset.

3.3.16 IRowset

IRowset is the base rowset interface. It provides method for fetching rows sequentially, getting the data from those rows, and managing rows.

IRowset requires **IAccessor** and **IRowsetInfo**. It is a mandatory interface on the rowset object.

Consumers use the methods in **IRowset** for all basic rowset operations, including fetching and releasing rows and getting column values.

When a consumer first gets an interface pointer on a rowset, usually its first step is to determine the rowset's capabilities using **IRowsetInfo::GetProperties**. This returns information about the interfaces exposed by the rowset as well as those capabilities of the rowset which do not show up as distinct interfaces, such as the maximum number of active rows and how many rows can have pending updates at the same time.

For most consumers, the next step is to determine the characteristics, or metadata, of the columns in the rowset. For this, they use either **IColumnsInfo** or **IColumnsRowset** for simple or extended column information, respectively. These interfaces are also available on prepared commands prior to execution, allowing advance planning.

The consumer determines which columns it needs, either from the metadata or on the basis of knowing the text command that generated the rowset. It determines the ordinals of the needed columns from the ordering of the column information returned by **IColumnsInfo** or from ordinals in the column metadata rowset returned by **IColumnsRowset**.

Some consumers do not use a command or do not want to browse the column information; they may know the name or property identifier for the columns

they want to use. They call **IColumnsInfo::MapColumnIDs** to retrieve the column ordinals.

The ordinals are used to specify a binding to a column. A binding is a structure that associates an element of the consumer's structure with a column. The binding can bind the column's data value, length, and status value.

A set of binding is gathered together in an accessor, which is created with **IAccessor::CreateAccessor**. An accessor can contain multiple bindings so that the data for multiple columns can be retrieve or set in a single call. The consumer can create several accessors to match different usage patterns in different parts of the application. It can create and release accessors at any time while the rowset remains in existence.

To fetch rows from the database, the consumer calls a method such as **IRowset::GetNextRows** or **IRowsetLocate::GetRowsAT**. To create and initialize a new row to be inserted into the data store, the consumer calls **IRowsetChange::InsertRow**.

The methods that fetch rows do not actually return data to the consumer. Instead, they return the handles to these rows and a local copy of the rows is stored in the rowset.

After the rows are returned, the consumer can access the data in the rows. The consumer calls **IRowset::GetData** and passes it the handle to a row, the handle to an accessor, and a pointer to a consumer-allocated buffer**.** **IRowset::GetData** converts the data (if it does not match the native provider storage) and returns the columns as specified in the bindings used to create the accessor. The consumer can call **IRowset::GetData** more than once for a row, using different accessors and buffers; therefore, the consumer can have multiple copies of the same data. For example, if a column contains a text document, the consumer might call **IRowset::GetData** with an accessor that binds the first 50 bytes of the document. When the user double-clicks on the displayed heading text, the consumer could then call **IRowset::GetData** with a different accessor to retrieve the entire document.
Data from variable-length columns may be treaded several ways. First, such columns can be bound to a finite section of the consumer's structure, which causes truncation when the length of the data exceeds the length of the buffer. The consumer can determine that truncation has occurred by checking whether the status is DBSTATUS_S_TRUNCATED. The returned length is always the true length in bytes, so the consumer also can determine how much data was truncated. Another way to obtain data from such column is by reference. For example, if a binary column is bound with a type indicator of DBTYPE_BYTES | DBTYPE_BYREF, the provider allocates memory for all of the data in the column and returns this memory to the consumer.

When the consumer is finished fetching or updating rows, it releases them with **IRowset::ReleaseRows**. This releases resources from the rowset's copy of the rows and makes room for new rows. The consumer can then repeat its cycle of fetching or creating rows and accessing the data in them.

When the consumer is done with the rowset, it calls **IAccessor::ReleaseAccessor** to release any accessors. It calls **IUnknown::Release** on all interfaces exposed by the rowset to release the rowset. When the rowset is released, it forces the release of any remaining

rows or accessors the consumer may hold. Such handle objects are subordinate to the rowset. That is, they do not take reference counts upon the rowset and cannot cause the rowset to linger beyond the point where all the interfaces for the rowset have been released. The rowset must clean up all such subordinate objects.

NOTE If the rowset was generated as a result of executing a command that contained output parameters and the provider populates output parameters when the rowset is released (that is, DBPROP_OUTPUTPARAMETERAVAILABILITY is DBPROP_OA_ATROWRELEASE), the memory for the output parameters bound when command was executed must be valid when the rowset is release. Not doing so is considered a serious programming error and likely will cause a crash.

Method	Description
AddRefRows	Adds a reference count to an existing row handle.
GetData	Retrieves data from the rowset's copy of the row.
GetNextRows	Fetches rows sequentially, remember the previous position.
ReleaseRows	Releases rows.
RestartPostition	Repositions the next fetch position to its initial position; that is, its position when the rowset was first created.

3.3.17 IRowsetInfo

IRowsetInfo provides information about a rowset. All rowsets must implement this interface.

When a consumer gets an interface pointer on a rowset, its first step usually is to determine the rowset's capabilities using **IUnknown::QueryInterface**. It may call **IRowsetInfo::GetProperties** to learn the properties of the rowset that do not show up as distinct interfaces, such as the maximum number of active rows and how many rows can have pending updates at the same time.

IRowsetInfo also provides methods for retrieving objects associated with the rowset. **IRowsetInfo::GetSpecification** gets the object (command or session) that created the rowset**. IRowsetInfo::GetReferencedRowset** gets the rowset that is referenced by a bookmark-valued column.

Method	Description
GetProperties	Returns the current setting of all properties supported by the rowset.
GetReferencedRowset	Returns an interface pointer to the rowset to which a bookmark applies.
GetSpecification	Returns an interface pointer on the object (command or session) that created the rowset.

3.3.18 IConnectionPointContainer

The **IConnectionPointContainer** interface supports connection points for connectable objects. When implemented on an object, makes the object connectable and expresses the existence of outgoing interface on the object.

Through this interface a client can locate a specific connection point for one IID. (This is an OLE, not an OLE DB interface.)

The consumers call **IConnectionPointContainer::EnumConnectionPoints** to create an enumerator object to iterate through all the connection points supported in the connectable object, one connection point per outgoing IID.

The connection point is a separate sub-object to avoid circular reference counting problems.

Method	Description
EnumConnectionPoints	Returns an object to enumerate all the connection points supported in the connectable object.
FindConnectionPoint	Returns a connection pointer to a specific IID.

3.3.19 IDBAsynchStatus

Consumers requiring asynchronous data source object initialization or asynchronous rowset generation or population can poll for status or cancel the asynchronous operation by requesting **IDBAsynchStatus** on the object being initialized, generated, or populated.

Method	Description
Abort	Cancels an asynchronously executing operation.
GetStatus	Returns the status of an asynchronously executing operation.

3.3.20 IRowsetChange

The methods in **IRowsetChange** are used to update the values of columns in existing rows, delete existing rows, and insert new rows. This interface requires IAccessor and IRowset.

Rowsets implement **IRowsetChange** if they support updating, deleting, or inserting rows. They are not required to support all three but must support all three but must support at least one of these operations to support **IRowsetChange**. The rowset reports which operations it supports through the DBPROP_UPDATABILITY property.

The consumer calls methods in **IRowsetChange** to modify rows as follows.

- **IRowsetChange::SetData** sets column data in an existing row.
- **IRowsetChange::DeleteRows** deletes existing rows.
- **IRowsetChange::InsertRow** creates a new row and sets initial values.

IRowsetChange::SetData and **IRowsetChange::InsertRow** require the use of an accessor. They can fail for a number of reasons. The most common of these is that new data values do not meet the schema or integrity constraints of the column. Furthermore, rowsets can have row-by-row and

column-by-column access permissions that override the general permissions of the table or column.

Method	Description
DeleteRows	Deletes rows.
InsertRow	Creates and initializes a new row.
SetData	Sets data in one or more columns in a row.

3.3.21 IRowsetFind

IRowsetFind is the interface that allows consumers to find a row within the rowset matching a specified value. Consumers call IRowsetFind to locate the next row in the rowset matching a specified value.

This interface is an optional interface on the rowset object. However, general consumers expect the rowset object to implement this interface, either natively or via OLE DB Services.

Method	Description
FindNextRow	Returns the next row matching the criteria.

3.3.22 IRowsetIndex

IRowsetIndex is the primary interface for exposing index functionality in OLE DB. For integrated indexes, **IRowsetIndex** is exposed on the base table rowset; otherwise, it is exposed on the index rowset.

The methods in **IRowsetIndex** are used to define a range of index entries to be read, to position at an index entry within the range, to fetch the index entry, and to access the contents of the index entry.

Method	Description
GetIndexInfo	Returns information about the index rowset capability.
Seek	Allows direct positioning at a key value within the current range.
SetRange	Restricts the set of row entries visible through calls to IRowset::GetNextRows and IRowsetIndex::Seek.

3.3.23 IRowsetLocate

IRowsetLocate is the interface for fetching arbitrary rows of a rowset. A rowset that does not implement this interface is a sequential rowset. **IRowsetLocate** is a prerequisite for **IRowsetScroll**.

When **IRowsetLocate** or one of its direct descendants is present on a rowset, column 0 is the bookmark for the rows. Reading this column will obtain a bookmark value that can be used to reposition to the same row.

Method	Description
Compare	Compares two bookmarks.
GetRowsAt	Fetches rows, starting with the row specified by an offset from a bookmark.
GetRowsByBookmark	Fetches the rows that match the specified bookmarks.
Hash	Returns hash values for the specified bookmark.

3.3.24 IRowsetRefresh

IRowsetRefresh is used to retrieve the values for rows that are currently visible to the transaction. **IRowsetRefresh** is optional. It is used to synchronize rows in the rowset with those in the data store. The primary uses of **IRowsetRefresh** are as follow:

- To implement optimistic concurrency.
- To perform row fix-up in a disconnected environment.

Method	Description
GetLastVisibleData	Gets the most recent data from either the visible data cache or the data store.
RefreshVisibleData	Retrieves the data values from the data store that are visible to the transaction for the specified rows.

3.3.25 IRowsetScroll

IRowsetScroll enables consumers to fetch rows at approximate positions in the rowset.

IRowsetScroll is an optional interface on the rowset object. However, general consumers expect the rowset object to implement this interface, either natively or via OLE DB Services. It is implemented for rowsets intended to work smoothly with long lists browsed directly by the user interface.

IRowsetScroll implies **IRowsetLocate.**

Method	Description
GetApproximatePosition	Gets the approximate position of a row corresponding to a specified bookmark.
GetRowsAtRatio	Fetches rows starting from a fractional position in the rowset.

3.3.26 IRowsetUpdate

IRowsetUpdate enables **consumers** to delay the transmission of the changes made with **IRowsetChange** to the data store. This interface also enables consumers to undo changes before transmission. This interface is optional.

Rowsets that **want** to allow delayed transmission of changes and to support undo capabilities implement IRowsetUpdate. Generally, this requires the

rowset to locally cache a copy of each row as it is changed with **IRowsetChange::SetData**.

The consumer calls the method in **IRowsetChange** to update, delete, and insert rows. The **rowset** buffers these changes. When the consumer is ready to transmit these changes to the data store, it calls **IRowsetUpdate::Update**. Before calling Update, the consumer can back out any changes with **IRowsetUpdate::Undo**.

Method	Description
GetOriginalData	Gets the data most recently fetched from or transmitted to the data store; does not get values based on pending changes.
GetPendingRows	Returns a list of rows with pending changes.
GetRowStatus	Returns the status of rows.
Undo	Undoes any changes made to a row since it was last fetched or since Update was called for it.
Update	Transmits any changes made to a row since it was last fetched or since Update was called for it.

3.3.27 IRowsetView

IRowsetView enables the **consumer** to create or apply a view to an existing rowset.

Consumers call **IRowsetView** to obtain a view object from a rowset on which to apply post-processing operations to an existing rowset. The consumer calls **IRowsetView::CreateView** to create a view object and calls methods on the view to apply post-processing operations to that view object. The consumer then calls **IViewChapter::OpenViewChapter** to retrieve a chapter handle that reflects the specified view conditions on the original rowset, or **IViewRowset::OpenViewRowset** to create a new rowset that reflects the specified view conditions. Any resources associated with the view opened as a chapter of an existing rowset can be cleaned up with a call to **IChapteredRowset::ReleaseChapter**.

Method	Description
CreateView	Creates a view object.
GetView	Returns a view object reflecting the conditions applied to the specified chapter.

4 Samples

This sample demonstrates the rowset programming and object model for an OLE DB consumer. It creates data source, session, and rowset objects; allows the user to display and navigate the rows in the rowset; and handles errors, if any. Command line switches determine whether an enumerator, class ID, user prompt, or connection string is used to create the data source object, whether or not a command is used to create the rowset, and so on.

4.1 OLE DB Consumer Application Examples in Microsoft Visal C++

4.1.1 Summary of Routines

The table below shows the structure of the routines in the sample. The # column is the ordinal number of the routine named in the **Routine** column; the **Routine** column gives a brief description of what the routine does. Some routines contain subroutines, which are shown indented within their containing routine. The ordinal number of the called subroutine is listed in the **Calls** column. Click any routine name in this table to display the actual routine.

Within a routine, click the "Summary of Routines" link in the block comment to display the following table. Click any linked interface name to display the interface topic in the OLE DB Programmer's Reference.

The table below shows the structure of the routines in the sample.

#	Calls	Routine
0		Preprocessor statements and prototypes—Necessary preprocessor definitions and declarations.
1		main—Begin the sample consumer application.
	2	myParseCommandLine
	3	myDisplayInstructions
	7	myCreateDataSource
	11	myCreateSession
	13	myCreateRowset
	16	myDisplayRowset

#	Calls	Routine
2		myParseCommandLine—Parses command line switches that control whether the sample creates the data source object with the Data Link UI or uses an enumerator and connection string; whether or not a command creates the rowset; whether to fetch a BLOB; and whether to display method call strings when reporting errors.
3		myDisplayInstructions—Displays instructions on how to use the sample based on the command line switches.
	5	myGetChar
4		myGetInputFromUser—Prompts the user for variable input, such as command text.
5		myGetChar—Gets a character from the keyboard.
6		myCreateEnumerator—Enumerates the providers on the user's machine and allows the user to select one.
	8	myDoInitialization
	21	myAddRowsetProperties
	16	myDisplayRowset
	4	myGetInputFromUser
7		myCreateDataSource—Creates a data source object, based on the user's previous specifications.
	6	myCreateEnumerator
	8	myDoInitialization
8		myDoInitialization—Sets the provider initialization properties and then initializes the data source object.
	10	myAddProperty
9		myGetProperty—Gets the value of the specified property.
10		myAddProperty—Adds a property to an array of properties.
11		myCreateSession—Creates a session object from a data source object.
12		myCreateSchemaRowset—Obtains the names of, and allows the user to select, tables supported by a provider from a table schema rowset.
	21	myAddRowsetProperties
	16	myDisplayRowset
13		myCreateRowset—Creates a rowset from a default table name, command, or user-specified table name.
	12	myCreateSchemaRowset
	21	myAddRowsetProperties
	24	myCreateCommand

#	Calls	Routine
	4	myGetInputFromUser
	25	myExecuteCommand
14		mySetupBindings—Uses the columns of the obtained rowset to specify an array of bindings. Bindings specify how data in columns should optionally be converted and stored in user variables.
	9	myGetProperty
15		myCreateAccessor—Creates an accessor, which specifies the bindings to use to fetch data from the rowset.
	14	mySetupBindings
16		myDisplayRowset—Displays data from a rowset and performs basic rowset navigation.
	9	myGetProperty
	23	myFindColumn
	15	myCreateAccessor
	22	myUpdateDisplaySize
	18	myDisplayColumnNames
	19	myDisplayRow
	17	myInteractWithRowset
	20	myFreeBindings
17		myInteractWithRowset—Prompts the user for rowset navigation directives (next, previous, and restart); allows the user to select a row; exits the sample program upon request.
	5	myGetChar
18		myDisplayColumnNames—Displays rowset column names.
19		myDisplayRow—Displays the data in a row of the rowset.
20		myFreeBindings—Frees memory allocated for bindings.
21		myAddRowsetProperties—Specifies optional properties that identify features the rowset should support.
	10	myAddProperty
22		myUpdateDisplaySize—Manages the size of the column displayed.
23		myFindColumn—Find the specified index column.
24		myCreateCommand—Creates a command object, if possible.
25		myExecuteCommand—Sets properties on a command object, executes the command, and returns a rowset object.

#	Calls	Routine
26		myHandleResult—Handles and displays error information.
	27	myDisplayErrorRecord
	28	myDisplayErrorInfo
27		myDisplayErrorRecord—Displays error information for a single error record.
	29	myGetSqlErrorInfo
28		myDisplayErrorInfo—Displays basic error information for an error object that doesn't support error records.
29		myGetSqlErrorInfo—Displays, if possible, SQL and native error information.

Sample OLE DB Consumer Application Routines

```
//--------------------------------------------------------------------
// @module PRSAMPLE.H
//--------------------------------------------------------------------


////////////////////////////////////////////////////////////////
// Includes
//  Summary of Routines
//
////////////////////////////////////////////////////////////////
#ifndef __PRSAMPLE_H__
#define __PRSAMPLE_H__

#include "oledb.h"        // OLE DB Header
#include "oledberr.h"     // OLE DB Errors

#include "msdasc.h"       // OLE DB Service Component header
#include "msdaguid.h"     // OLE DB Root Enumerator
#include "msdasql.h"      // MSDASQL - Default provider

#include <stdio.h>        // input and output functions
#include <conio.h>        // getch, putch
#include <locale.h>       // setlocale

////////////////////////////////////////////////////////////////
// Globals
//
////////////////////////////////////////////////////////////////
extern DWORD g_dwFlags;


////////////////////////////////////////////////////////////////
// Defines
//
////////////////////////////////////////////////////////////////
#define __LONGSTRING(string) L##string
#define LONGSTRING(string) __LONGSTRING(string)

// Goes to CLEANUP on Failure_
```

```
#define CHECK_HR(hr)      \
    if(FAILED(hr))        \
        goto CLEANUP

// Goes to CLEANUP on Failure, and displays any ErrorInfo
#define XCHECK_HR(hr)                                          \
{                                                  \
    if( g_dwFlags & DISPLAY_METHODCALLS )                      \
        fwprintf(stderr, LONGSTRING(#hr) L"\n");               \
    if(FAILED(myHandleResult(hr, LONGSTRING(__FILE__), __LINE__)))   \
        goto CLEANUP;                                          \
}
#define CHECK_MEMORY(hr, pv)   \
{                          \
    if(!pv)                  \
    {                        \
        hr = E_OUTOFMEMORY;     \
        CHECK_HR(hr);           \
    }                        \
}


#define MAX_COL_SIZE        5000
#define MAX_NAME_LEN         256

#define MAX_ROWS            10
#define MAX_DISPLAY_SIZE     20
#define MIN_DISPLAY_SIZE     3

// ROUNDUP on all platforms pointers must be aligned properly
#define ROUNDUP_AMOUNT          8
#define ROUNDUP_(size,amount) (((ULONG)(size)+((amount)-1))&~((amount)-1))
#define ROUNDUP(size)           ROUNDUP_(size, ROUNDUP_AMOUNT)

enum
{
    // Connecting
    USE_PROMPTDATASOURCE     = 0x0001,
    USE_ENUMERATOR           = 0x0002,

    // Rowset
    USE_COMMAND              = 0x0010,

    // Storage objects
    USE_ISEQSTREAM           = 0x0100,

    // Display options
    DISPLAY_METHODCALLS      = 0x1000,
    DISPLAY_INSTRUCTIONS     = 0x2000,
};


////////////////////////////////////////////////////////////////////////////
// Function prototypes
//
////////////////////////////////////////////////////////////////////////////
// Main
BOOL   myParseCommandLine();
void   myDisplayInstructions();
```

```
BOOL   myGetInputFromUser(LPWSTR pwszInput, LPCWSTR pwszFmt, ...);
CHAR   myGetChar();

// Enumerator
HRESULT myCreateEnumerator(REFCLSID clsidEnumerator, CLSID* pCLSID);

// Data source
HRESULT myCreateDataSource(IUnknown** ppUnkDataSource);
HRESULT myDoInitialization(IUnknown* pIUnknown);
HRESULT myGetProperty(IUnknown* pIUnknown, REFIID riid, DBPROPID
                      dwPropertyID, REFGUID guidPropertySet, BOOL*
                      pbValue);
void myAddProperty(DBPROP* pProp, DBPROPID dwPropertyID, VARTYPE vtType =
                   VT_BOOL, LONG lValue = VARIANT_TRUE, DBPROPOPTIONS
                   dwOptions =      DBPROPOPTIONS_OPTIONAL);

// Session
HRESULT myCreateSession(IUnknown* pUnkDataSource, IUnknown**
                        ppUnkSession);
HRESULT myCreateSchemaRowset(GUID guidSchema, IUnknown* pUnkSession,
                             ULONG cchBuffer, LPWSTR pwszBuffer);

// Command
HRESULT myCreateCommand(IUnknown* pUnkSession, IUnknown** ppUnkCommand);
HRESULT        myExecuteCommand(IUnknown*        pUnkCommand,        WCHAR*
pwszCommandText,
                          ULONG cPropSets, DBPROPSET* rgPropSets,
                          IUnknown** ppUnkRowset);

// Rowset
HRESULT myCreateRowset(IUnknown* pUnkSession, IUnknown** ppUnkRowset);
HRESULT mySetupBindings(IUnknown* pUnkRowset, ULONG* pcBindings,
                        DBBINDING** prgBindings, ULONG* pcbRowSize);
HRESULT myCreateAccessor(IUnknown* pUnkRowset, HACCESSOR* phAccessor,
                         ULONG* pcBindings, DBBINDING** prgBindings,
                         ULONG* pcbRowSize);

HRESULT  myDisplayRowset(IUnknown* pUnkRowset, LPCWSTR pwszColToReturn,
                         ULONG cchBuffer, LPWSTR pwszBuffer);
HRESULT myDisplayColumnNames(IUnknown* pUnkRowset, ULONG* rgDispSize);
HRESULT myDisplayRow(ULONG iRow, ULONG cBindings, DBBINDING* rgBindings,
                     void* pData, ULONG * rgDispSize);

HRESULT myInteractWithRowset(IRowset* pIRowset, LONG* pcRows, DBCOUNTITEM
                             cRowsObtained, BOOL fCanFetchBackwards,
                             void* pData, ULONG cbRowSize, DBBINDING*
                             pBinding, ULONG cchBuffer, LPWSTR
                             pwszBuffer);
HRESULT myFindColumn(IUnknown * pUnkRowset, LPCWSTR pwszName, LONG*
                     plIndex);
HRESULT myUpdateDisplaySize(ULONG cBindings, DBBINDING* rgBindings, void*
                            pData, ULONG* rgDispSize);
void myFreeBindings(ULONG cBindings, DBBINDING* rgBindings);
void myAddRowsetProperties(DBPROPSET* pPropSet, ULONG cProperties,
                           DBPROP* rgProperties);

// Error
HRESULT myHandleResult(HRESULT hrReturned, LPCWSTR pwszFile, ULONG
                       ulLine);
```

```
HRESULT myDisplayErrorRecord(HRESULT hrReturned, ULONG iRecord,
                             IErrorRecords* pIErrorRecords, LPCWSTR
                             pwszFile, ULONG ulLine);
HRESULT myDisplayErrorInfo(HRESULT hrReturned, IErrorInfo* pIErrorInfo,
                           LPCWSTR pwszFile, ULONG ulLine);
HRESULT myGetSqlErrorInfo(ULONG iRecord, IErrorRecords* pIErrorRecords,
                          BSTR* pBstr, LONG* plNativeError);


#endif     // __PRSAMPLE_H__



//--------------------------------------------------------------------
// @module MAIN.CPP
//--------------------------------------------------------------------


////////////////////////////////////////////////////////////////
// Includes
//
////////////////////////////////////////////////////////////////
#define DBINITCONSTANTS   // Store all OLE DB consts inside
                          // this .obj file.
#include "prsample.h"     // Programmer's Reference Sample includes


////////////////////////////////////////////////////////////////
// Globals
//
////////////////////////////////////////////////////////////////
DWORD g_dwFlags = USE_PROMPTDATASOURCE | DISPLAY_METHODCALLS;


////////////////////////////////////////////////////////////////
// main
// Summary of Routines
//
// This is a simple OLE DB application that will display a
// rowset and will allow basic navigation of that rowset by the
// user.
//
// In the sample, functions that begin with 'my' are implemented
// in the sample code; all other functions are either OLE DB
// methods or standard system methods. In addition, two
// macros are used repeatedly throughout the sample:
//  - CHECK_HR(hr) - this macro goes to the CLEANUP label if
//    FAILED(hr), where hr is usually a method call.
//  - XCHECK_HR(hr) - this macro prints the string
//    representation of hr to stderr and if FAILED(hr), attempts
//    to obtain and display any extended error information
//    posted by the last method call and then jumps to the CLEANUP
//    label.
//
// This is the entry point for the sample. This function will:
//  - parse command line arguments passed to the sample.
//  - display appropriate instructions based on these arguments.
//  - create an OLE DB data source object for a user-chosen
```

```
//     provider.
//  - create an OLE DB session object from the provider's
//     data source object.
//  - create an OLE DB rowset object, over a table specified by
//     the user, from the provider's session object.
//  - display the rowset data and will allow the user to
//     navigate over the rowset.
//
////////////////////////////////////////////////////////////////
int main()
{
    HRESULT      hr;
    IUnknown *   pUnkDataSource   = NULL;
    IUnknown *   pUnkSession      = NULL;
    IUnknown *   pUnkRowset       = NULL;

    // Parse command line arguments, if any; this will update
    // the value of g_dwFlags as appropriate for the arguments.
   if( !myParseCommandLine() )
        return EXIT_FAILURE;

    // Display instructions for the given command line arguments.
     myDisplayInstructions() ;

    // Initialize OLE
    hr = CoInitialize(NULL);
    if( FAILED(hr) )
        return EXIT_FAILURE;

    // Create the data source object using the OLE DB service components.
      CHECK_HR(hr = myCreateDataSource(&pUnkDataSource));

     // Create a session object from the data source object.
      CHECK_HR(hr = myCreateSession(pUnkDataSource, &pUnkSession));

     // Create a rowset object from the session object, either directly
    // from the session or through a command object.
      CHECK_HR(hr = myCreateRowset(pUnkSession, &pUnkRowset));

    // Display the rowset object data to the user.
    CHECK_HR(hr = myDisplayRowset(pUnkRowset, NULL, 0, NULL));

CLEANUP:
    if( pUnkRowset )
        pUnkRowset->Release();
    if( pUnkSession )
        pUnkSession->Release();
    if( pUnkDataSource )
        pUnkDataSource->Release();

    CoUninitialize();

    if( FAILED(hr) )
        return EXIT_FAILURE;

    return EXIT_SUCCESS;
}
```

```
//////////////////////////////////////////////////////////////////
// myParseCommandLine
//  Summary of Routines
//
// This function parses the application's command line arguments
// and sets the appropriate bits in g_dwFlags. If an invalid
// argument is encountered, a usage message is displayed and
// the function returns FALSE; otherwise, TRUE is returned.
//
//////////////////////////////////////////////////////////////////
BOOL myParseCommandLine
    (
    void
    )
{
    int     iArg;
    CHAR *   psz;

    // Set the locale for all C run-time functions.
    setlocale(LC_ALL, ".ACP");

    // Go through each command line argument and set the appropriate
    // bits in g_dwFlags, depending on the chosen options.
    for( iArg = 1; iArg < __argc; iArg++ )
    {
        // Inspect the current argument string.
        psz = __argv[iArg];

        // Valid options begin with '-' or '/'.
        if( psz[0] == '-' || psz[0] == '/' )
        {
            // The next character is the option.
            switch( tolower(psz[1]) )
            {
            case 'u':
                // Use the service components UI to prompt for and create
                // the data source object; the enumerator is not used.
                g_dwFlags |= USE_PROMPTDATASOURCE;
                g_dwFlags &= ~USE_ENUMERATOR;
                continue;
            case 'e':
                // Use the enumerator to select the provider, and then use
                // IDataInitialize to create the data source object.
                // Don't use the UI to prompt for the data source.
                g_dwFlags |= USE_ENUMERATOR;
                g_dwFlags &= ~USE_PROMPTDATASOURCE;
                continue;
            case 'c':
                // Use ICommand instead of IOpenRowset.
                g_dwFlags |= USE_COMMAND;
                continue;
            case 'b':
                // Use ISequentialStream to fetch BLOB column data.
                g_dwFlags |= USE_ISEQSTREAM;
                continue;
            case 'n':
                // Don't display method call strings as part of
                // the extended error checking macro.
```

```
                g_dwFlags &= ~DISPLAY_METHODCALLS;
                continue;
            }
        }

        // Invalid argument; show the usage flags to the user.
        fprintf(stderr, "Usage: %s [-u] [-e] [-c] [-b] [-n]\n\nWhere:\n\t" \
            "u = Use the Microsoft Data Links UI " \
                "to create the DataSource\n\t" \
            "e = Use the Enumerator and IDataInitialize " \
                "to create the DataSource\n\t" \
            "c = Use ICommand instead of IOpenRowset to create the Rowset\n\t" \
            "b = Use ISequentialStream for BLOB columns\n\t" \
            "n = Don't display method call strings\n",
            __argv[0]);

        return FALSE;
    }

    return TRUE;
}


//////////////////////////////////////////////////////////////////////
// myDisplayInstructions
// Summary of Routines
//
// This function asks the user whether they would like
// instructions displayed for the application. If so, it
// displays the instructions appropriate to the flags set
// in g_dwFlags.
//
//////////////////////////////////////////////////////////////////////
void myDisplayInstructions
    (
    void
    )
{
    CHAR ch;

    // Display header and ask the user if they want instructions.
    printf("\nOLE DB Programmer's Reference Sample\n" \
            "==================================\n\n");
    printf("Display instructions [Y or N]? ");
    do
    {
     ch =  myGetChar();
     }
    while( ch != 'y' && ch != 'n' );
    printf("%c\n\n", ch);

    // No instructions, so we're done.
    if( ch == 'n' )
        return;

    // Display basic instructions.
    printf("This application is a simple OLE DB sample that will display\n" \
            "a rowset and will allow basic navigation of that rowset by\n" \
            "the user. The application will perform the following\n" \
```

```
            "steps:\n\n");

    // Display data source creation instructions.
    if( g_dwFlags & USE_PROMPTDATASOURCE )
    {
        printf(" - Creates a data source object through the Microsoft
                Data\n" \ "   Links UI. This allows the user to select
                the OLE DB\n" \ "   provider to use and to set connection
                properties.\n");
    }
    else
    {
        printf(" - Creates a data source object through
                IDataInitialize::\n" \
                "   CreateDBInstance, which allows the OLE DB service\n" \
                "   component manager to add additional functionality to\n"\
                "   the provider as requested. The user will select the\n" \
                "   provider to use from a rowset obtained from the OLE
                DB\n" \ "   enumerator.\n");
    }

    // Display session creation and table-selection instructions.
    printf(" - Creates a session object from the data source object.\n");
    printf(" - If the provider supports the schema rowset interface,\n" \
             "   creates a TABLES schema rowset and allows the user to\n" \
             "   select a table name from this rowset.\n");

    // Display rowset creation instructions
    if( g_dwFlags & USE_COMMAND )
    {
        printf(" - Creates a cxommand object from the session object and\n" \
                "   allows the user to specify command text for this Command,\n" \
                "   then executes the command to create the final rowset.\n");
    }
    else
    {
        printf(" - Creates the final rowset over the table specified by the\n" \
                "   user.\n");
    }

    printf(" - Displays this rowset and allows the user to perform basic\n" \
             "   navigation of that rowset.\n\n");

    // Wait for the user to press a key before continuing.
    printf("Press a key to continue...");
myGetChar();
    printf("\n\n");
}



//////////////////////////////////////////////////////////////////////
// myGetInputFromUser
//  Summary of Routines
//
// This function prompts the user with the contents of pwszFmt
// and any accompanying variable arguments and then gets a string
// as input from the user. If the string is non-empty, it is
// copied into pwszInput and the function returns TRUE;
```

```
// otherwise, this function returns FALSE.
//
////////////////////////////////////////////////////////////////
BOOL myGetInputFromUser
    (
    LPWSTR    pwszInput,
    LPCWSTR   pwszFmt,
    ...
    )
{
    va_list   vargs;
    WCHAR     wszBuffer[MAX_NAME_LEN + 1]     = {0};

    // Create the string with variable arguments...
    va_start(vargs, pwszFmt);
    _vsnwprintf(wszBuffer, MAX_NAME_LEN, pwszFmt, vargs);
    va_end(vargs);

    // Output the string...
    wprintf(wszBuffer);

    // Now get the Input from the user....
    _getws(wszBuffer);
    if( wszBuffer[0] )
    {
        wcscpy(pwszInput, wszBuffer);
        return TRUE;
    }

    return FALSE;
}


////////////////////////////////////////////////////////////////
// myGetChar
//  Summary of Routines
//
// This function gets a character from the keyboard and
// converts it to lowercase before returning it.
//
////////////////////////////////////////////////////////////////
CHAR myGetChar
    (
    void
    )
{
    CHAR ch;

    // Get a character from the keyboard.
    ch = _getch();

    // Re-read for the actual key value if necessary.
    if( !ch || ch == 0xE0 )
        ch = _getch();

    return tolower(ch);
}
```

```
//--------------------------------------------------------------------
// @module ENUM.CPP
//--------------------------------------------------------------------


////////////////////////////////////////////////////////////////
// Includes
//
////////////////////////////////////////////////////////////////
#include "prsample.h"          // Programmer's Reference Sample includes



////////////////////////////////////////////////////////////////
// myCreateEnumerator
//  Summary of Routines
//
// This function creates an enumerator, obtains a sources rowset
// from it, displays the rowset to the user, and allows the user
// to specify the ProgID of a provider. The CLSID that matches
// this ProgID is retuned to the caller in *pCLSID.
//
////////////////////////////////////////////////////////////////
HRESULT myCreateEnumerator
    (
    REFCLSID          clsidEnumerator,
    CLSID *           pCLSID
    )
{
    HRESULT           hr;
    IUnknown *        pIUnkEnumerator             = NULL;
    ISourcesRowset *   pISourcesRowset             = NULL;
    IRowset *         pIRowset                    = NULL;
    IDBInitialize *    pIDBInitialize              = NULL;
    WCHAR             wszProgID[MAX_NAME_LEN + 1]   = {0};

    const ULONG       cProperties                 = 2;
    DBPROP            rgProperties[cProperties];
    DBPROPSET         rgPropSets[1];

    // Create the enumerator object. We ask for IUnknown when creating
    // the enumerator because some enumerators may require initialization
    // before we can obtain a sources rowset from the enumerator. This is
    // indicated by whether the enumerator object exposes IDBInitialize
    // or not. (We don't want to ask for IDBInitialize, since enumerators
    // that don't require initialization will cause the CoCreateInstance
    // to fail.)
    XCHECK_HR(hr = CoCreateInstance(
                clsidEnumerator,                // clsid -- enumerator
                NULL,                           // pUnkOuter
                CLSCTX_INPROC_SERVER,           // dwClsContext
                IID_IUnknown,                   // riid
                (void**)&pIUnkEnumerator        // ppvObj
                ));

    // If the enumerator exposes IDBInitialize, we need to initialize it.
    // See IDBInitialize
```

```
    if( SUCCEEDED(hr = pIUnkEnumerator->QueryInterface(IID_IDBInitialize,
            (void**)&pIDBInitialize)) )
    {
    CHECK_HR(hr = myDoInitialization(pIUnkEnumerator));
    }

    // Set properties on the rowset, to request additional functionality.
    myAddRowsetProperties(rgPropSets, cProperties, rgProperties);

    // Obtain a sources rowset from the enumerator. This rowset contains
    // all of the OLE DB providers that this enumerator is able to list.
    XCHECK_HR(hr = pIUnkEnumerator->QueryInterface(IID_ISourcesRowset,
            (void**)&pISourcesRowset));
    XCHECK_HR(hr = pISourcesRowset->GetSourcesRowset(
            NULL,                       // pUnkOuter
            IID_IRowset,                // riid
            1,                          // cPropSets
            rgPropSets,                 // rgPropSets
            (IUnknown**)&pIRowset       // ppRowset
            ));

    // Display the rowset to the user. This will allow the user to
    // perform basic navigation of the rowset and will allow the user
    // to select a row containing a desired provider.
    CHECK_HR(hr = myDisplayRowset(pIRowset,
            L"SOURCES_NAME", MAX_NAME_LEN, wszProgID));

    // Obtain the ProgID for the provider to use from the user.
    // The default value for this is the value of the SOURCES_NAME
    // column in the row selected by the user previously.
myGetInputFromUser(wszProgID, L"\nType the ProgID of a provider"
            L" to use [Enter = `%s`]: ", wszProgID);
    XCHECK_HR(hr = CLSIDFromProgID(wszProgID, pCLSID));

CLEANUP:
    if( pIUnkEnumerator )
        pIUnkEnumerator->Release();
    if( pISourcesRowset )
        pISourcesRowset->Release();
    if( pIRowset )
        pIRowset->Release();
    if( pIDBInitialize )
        pIDBInitialize->Release();
    return hr;
}



//--------------------------------------------------------------------
// @module DATASOURCE.CPP
//--------------------------------------------------------------------


////////////////////////////////////////////////////////////////////
// Includes
//
////////////////////////////////////////////////////////////////////
```

```
#include "prsample.h"          // Programmer's Reference Sample includes


//////////////////////////////////////////////////////////////
// myCreateDataSource
// Summary of Routines
//
// This function creates an OLE DB data source object for a
// provider selected by the user, sets initialization properties
// for the data source, and initializes the data source. The
// function returns a pointer to the data source object's
// IUnknown in *ppUnkDataSource.
//
//////////////////////////////////////////////////////////////
HRESULT myCreateDataSource
    (
    IUnknown **            ppUnkDataSource
    )
{
    HRESULT                hr;
    IDataInitialize *      pIDataInitialize      = NULL;
    IDBPromptInitialize *  pIDBPromptInitialize  = NULL;
    IDBInitialize *        pIDBInitialize        = NULL;
    CLSID                  clsid                 = CLSID_MSDASQL;

    // Use the Microsoft Data Links UI to create the data source
    // object. This will allow the user to select the provider
    // to connect to and to set the initialization properties
    // for the data source object, which will be created by the
    // Data Links UI.
    if( g_dwFlags & USE_PROMPTDATASOURCE )
    {
        // Create the Data Links UI object, and obtain the
        // IDBPromptInitialize interface from it.
           XCHECK_HR(hr = CoCreateInstance(
                    CLSID_DataLinks,              // clsid -- Data Links UI
                    NULL,                         // pUnkOuter
                    CLSCTX_INPROC_SERVER,         // dwClsContext
                    IID_IDBPromptInitialize,      // riid
                    (void**)&pIDBPromptInitialize    // ppvObj
                    ));

        // Invoke the Data Links UI to allow the user to select
        // the provider and set initialization properties for
        // the data source object that this will create.
            XCHECK_HR(hr = pIDBPromptInitialize->PromptDataSource(
                    NULL,                         // pUnkOuter
                    GetDesktopWindow(),           // hWndParent
                    DBPROMPTOPTIONS_PROPERTYSHEET,    // dwPromptOptions
                    0,                            // cSourceTypeFilter
                    NULL,                         // rgSourceTypeFilter
                    NULL,                         // pwszszzProviderFilter
                    IID_IDBInitialize,            // riid
                    (IUnknown**)&pIDBInitialize      // ppDataSource
                    ));

        // We've obtained a data source object from the Data Links UI. This
        // object has had its initialization properties set, so all we
```

```
        // need to do is Initialize it.
        XCHECK_HR(hr = pIDBInitialize->Initialize());
    }
    // We are not using the Data Links UI to create the data source
    // object. Instead, we will enumerate the providers installed on this
    // system through the OLE DB enumerator and will allow the user to
    // select the ProgID of the provider for which we will create a
    // data source object.
    else
    {
        // Use the OLE DB enumerator to obtain a rowset of installed
        // providers, and then allow the user to select a provider from
        // this rowset.
    CHECK_HR(hr =
       myCreateEnumerator(CLSID_OLEDB_ENUMERATOR, &clsid));

        // We will create the data source object through the OLE DB service
        // component IDataInitialize interface, so we need to create an
        // instance of the data initialization object.
                XCHECK_HR(hr = CoCreateInstance(
                   CLSID_MSDAINITIALIZE,        // clsid -- data initialize
                   NULL,                        // pUnkOuter
                   CLSCTX_INPROC_SERVER,        // dwClsContext
                   IID_IDataInitialize,         // riid
                   (void**)&pIDataInitialize    // ppvObj
                   ));

        // Use IDataInitialize::CreateDBInstance to create an uninitialized
        // data source object for the chosen provider. By using this
        // service component method, the service component manager can
        // provide additional functionality beyond what is natively
        // supported by the provider if the consumer requests that
        // functionality.
                XCHECK_HR(hr = pIDataInitialize->CreateDBInstance(
                   clsid,                       // clsid -- provider
                   NULL,                        // pUnkOuter
                   CLSCTX_INPROC_SERVER,        // dwClsContext
                   NULL,                        // pwszReserved
                   IID_IDBInitialize,           // riid
                   (IUnknown**)&pIDBInitialize  // ppDataSource
                   ));

        // Initialize the data source object by setting any required
        // initialization properties and calling IDBInitialize::Initialize.
        CHECK_HR(hr = myDoInitialization(pIDBInitialize));
    }

CLEANUP:
    *ppUnkDataSource = pIDBInitialize;
    if( pIDataInitialize )
        pIDataInitialize->Release();
    if( pIDBPromptInitialize )
        pIDBPromptInitialize->Release();
    return hr;
}


//////////////////////////////////////////////////////////////////////
```

```
// myDoInitialization
// Summary of Routines
//
// This function sets initialization properties that tell the
// provider to prompt the user for any information required to
// initialize the provider and then calls the provider's
// initialization function.
//
//////////////////////////////////////////////////////////////
HRESULT myDoInitialization
    (
    IUnknown *        pIUnknown
    )
{
    HRESULT           hr;
    IDBInitialize *   pIDBInitialize   = NULL;
    IDBProperties *   pIDBProperties   = NULL;
    HWND              hWnd             = GetDesktopWindow();

    const ULONG       cProperties      = 2;
    DBPROP            rgProperties[cProperties];
    DBPROPSET         rgPropSets[1];

    // To initialize the data source object, most providers require
    // some initialization properties to be set by the consumer. For
    // example, these might include the data source to connect to and the
    // user ID and password to use to establish identity. We will ask the
    // provider to prompt the user for this required information by
    // setting the following properties:

  myAddProperty(&rgProperties[0],DBPROP_INIT_PROMPT,VT_I2,DBPR
  OMPT_COMPLETE);
  myAddProperty(&rgProperties[1],DBPROP_INIT_HWND,
  VT_I4,(LONG)hWnd);

     rgPropSets[0].rgProperties      = rgProperties;
    rgPropSets[0].cProperties       = cProperties;
    rgPropSets[0].guidPropertySet   = DBPROPSET_DBINIT;

    // Obtain the needed interfaces.
        XCHECK_HR(hr = pIUnknown->QueryInterface(IID_IDBProperties,
                (void**)&pIDBProperties));
    XCHECK_HR(hr = pIUnknown->QueryInterface(IID_IDBInitialize,
                (void**)&pIDBInitialize));

    // If a provider requires initialization properties, it must support
    // the properties that we are setting (_PROMPT and _HWND). However,
    // some providers do not need initialization properties and may
    // therefore not support the _PROMPT and _HWND properties. Because of
    // this, we will not check the return value from SetProperties.
    hr = pIDBProperties->SetProperties(1, rgPropSets);

    // Now that we've set our properties, initialize the provider.
    XCHECK_HR(hr = pIDBInitialize->Initialize());

CLEANUP:
    if( pIDBProperties )
```

```
        pIDBProperties->Release();
    if( pIDBInitialize )
        pIDBInitialize->Release();
    return hr;
}



//////////////////////////////////////////////////////////////
// myGetProperty
// Summary of Routines
//
// This function gets the BOOL value for the specified property
// and returns the result in *pbValue.
//
//////////////////////////////////////////////////////////////
HRESULT myGetProperty
    (
    IUnknown *          pIUnknown,
    REFIID              riid,
    DBPROPID            dwPropertyID,
    REFGUID             guidPropertySet,
    BOOL *              pbValue
    )
{
    HRESULT             hr;
    DBPROPID            rgPropertyIDs[1];
    DBPROPIDSET         rgPropertyIDSets[1];

    ULONG               cPropSets       = 0;
    DBPROPSET *         rgPropSets      = NULL;

    IDBProperties *     pIDBProperties  = NULL;
    ISessionProperties * pISesProps     = NULL;
    ICommandProperties * pICmdProps     = NULL;
    IRowsetInfo *       pIRowsetInfo    = NULL;

    // Initialize the output value
    *pbValue = FALSE;

    // Set up the property ID array
    rgPropertyIDs[0] = dwPropertyID;

    // Set up the Property ID Set
    rgPropertyIDSets[0].rgPropertyIDs    = rgPropertyIDs;
    rgPropertyIDSets[0].cPropertyIDs     = 1;
    rgPropertyIDSets[0].guidPropertySet  = guidPropertySet;

    // Get the property value for this property from the provider, but
    // don't try to display extended error information, since this may
    // not be a supported property. A failure is, in fact, expected if
    // the property is not supported.
      if( riid == IID_IDBProperties )
    {
        XCHECK_HR(hr = pIUnknown->QueryInterface(IID_IDBProperties,
                    (void**)&pIDBProperties));
        CHECK_HR(hr = pIDBProperties->GetProperties(
                    1,                          // cPropertyIDSets
                    rgPropertyIDSets,           // rgPropertyIDSets
```

```
                    &cPropSets,                 // pcPropSets
                    &rgPropSets                 // prgPropSets
                    ));
    }
    else if( riid == IID_ISessionProperties )
    {
        XCHECK_HR(hr = pIUnknown->QueryInterface(IID_ISessionProperties,
                    (void**)&pISesProps));
        CHECK_HR(hr = pISesProps->GetProperties(
                    1,                          // cPropertyIDSets
                    rgPropertyIDSets,           // rgPropertyIDSets
                    &cPropSets,                 // pcPropSets
                    &rgPropSets                 // prgPropSets
                    ));
    }
    else if( riid == IID_ICommandProperties )
    {
        XCHECK_HR(hr = pIUnknown->QueryInterface(IID_ICommandProperties,
                    (void**)&pICmdProps));
        CHECK_HR(hr = pICmdProps->GetProperties(
                    1,                          // cPropertyIDSets
                    rgPropertyIDSets,           // rgPropertyIDSets
                    &cPropSets,                 // pcPropSets
                    &rgPropSets                 // prgPropSets
                    ));
    }
    else
    {
        XCHECK_HR(hr = pIUnknown->QueryInterface(IID_IRowsetInfo,
                    (void**)&pIRowsetInfo));
        CHECK_HR(hr = pIRowsetInfo->GetProperties(
                    1,                          // cPropertyIDSets
                    rgPropertyIDSets,           // rgPropertyIDSets
                    &cPropSets,                 // pcPropSets
                    &rgPropSets                 // prgPropSets
                    ));
    }

    // Return the value for this property to the caller if
    // it's a VT_BOOL type value, as expected.
    if( V_VT(&rgPropSets[0].rgProperties[0].vValue) == VT_BOOL )
        *pbValue = V_BOOL(&rgPropSets[0].rgProperties[0].vValue);

CLEANUP:
    if( rgPropSets )
    {
        CoTaskMemFree(rgPropSets[0].rgProperties);
        CoTaskMemFree(rgPropSets);
    }
    if( pIDBProperties )
        pIDBProperties->Release();
    if( pISesProps )
        pISesProps->Release();
    if( pICmdProps )
        pICmdProps->Release();
    if( pIRowsetInfo )
        pIRowsetInfo->Release();
    return hr;
}
```

```
//////////////////////////////////////////////////////////////////
// myAddProperty
// Summary of Routines
//
// This function initializes the property structure pProp.
//
//////////////////////////////////////////////////////////////////
void myAddProperty
    (
    DBPROP *          pProp,
    DBPROPID          dwPropertyID,
    VARTYPE           vtType,
    LONG              lValue,
    DBPROPOPTIONS     dwOptions
    )
{
    // Set up the property structure.
    pProp->dwPropertyID    = dwPropertyID;
    pProp->dwOptions       = dwOptions;
    pProp->dwStatus        = DBPROPSTATUS_OK;
    pProp->colid           = DB_NULLID;
    V_VT(&pProp->vValue)   = vtType;

    // Since VARIANT data is a union, we can place the value in any
    // member (except for VT_DECIMAL, which is a union with the whole
    // VARIANT structure -- but we know we're not passing VT_DECIMAL).
    V_I4(&pProp->vValue)     = lValue;
}



//--------------------------------------------------------------------
// @module SESSION.CPP
//--------------------------------------------------------------------


//////////////////////////////////////////////////////////////////
// Includes
//
//////////////////////////////////////////////////////////////////
#include "prsample.h"      // Programmer's Reference Sample includes



//////////////////////////////////////////////////////////////////
// myCreateSession
// Summary of Routines
//
// Create an OLE DB session object from the given data source
// object. The IDBCreateSession interface is mandatory, so this
// is a simple operation.
//
//////////////////////////////////////////////////////////////////
HRESULT     myCreateSession
    (
    IUnknown *          pUnkDataSource,
```

```
    IUnknown **         ppUnkSession
    )
{
    HRESULT             hr;
    IDBCreateSession *   pIDBCreateSession         = NULL;

    //Create a session object from a data source object
      XCHECK_HR(hr = pUnkDataSource->QueryInterface(
                IID_IDBCreateSession, (void**)&pIDBCreateSession));
    XCHECK_HR(hr = pIDBCreateSession->CreateSession(
                NULL,                           // pUnkOuter
                IID_IOpenRowset,                    // riid
                ppUnkSession                      // ppSession
                ));

CLEANUP:
    if( pIDBCreateSession )
        pIDBCreateSession->Release();
    return hr;
}


//////////////////////////////////////////////////////////////////
// myCreateSchemaRowset
// Summary of Routines
//
// If the provider supports IDBSchemaRowset, this function will
// obtain the tables schema rowset, will display this rowset to
// the user, and will allow the user to select a row in the
// rowset containing the name of a table of interest.
//
//////////////////////////////////////////////////////////////////
HRESULT     myCreateSchemaRowset
    (
    GUID              guidSchema,
    IUnknown *          pUnkSession,
    ULONG              cchBuffer,
    LPWSTR              pwszBuffer
    )
{
    HRESULT             hr                = S_OK;
    IDBSchemaRowset *   pIDBSchemaRowset   = NULL;
    IUnknown *          pUnkRowset         = NULL;

    const ULONG         cProperties        = 2;
    DBPROP              rgProperties[cProperties];
    DBPROPSET           rgPropSets[1];

    // Attempt to obtain the IDBSchemaRowset interface on the session
    // object. This is not a mandatory interface; if it is not supported,
    // we are done.
  CHECK_HR(pUnkSession->QueryInterface(
                    IID_IDBSchemaRowset, (void**)&pIDBSchemaRowset));

    // Set properties on the rowset, to request additional functionality.
    myAddRowsetProperties(rgPropSets, cProperties, rgProperties);

    // Get the requested schema rowset. If IDBSchemaRowset is supported,
```

```
    // the following schema rowsets are required to be supported:
    //  BSCHEMA_TABLES,DBSCHEMA_COLUMNS,and
DBSCHEMA_PROVIDERTYPES
    // We know that we will be asking for one of these, so it is not
    // necessary to call IDBSchemaRowset::GetSchemas in this case.
       XCHECK_HR(hr = pIDBSchemaRowset->GetRowset(
                NULL,                       // pUnkOuter
                guidSchema,                    // guidSchema
                0,                       // cRestrictions
                NULL,                       // rgRestrictions
                IID_IRowset,                   // riid
                1,                       // cPropSets
                rgPropSets,                    // rgPropSets
                &pUnkRowset                    // ppRowset
                ));

    // Display the rowset to the user. This will allow the user to
    // perform basic navigation of the rowset and will allow the user
    // to select a row containing a desired table name (taken from the
    // TABLE_NAME column).
    CHECK_HR(hr = myDisplayRowset(pUnkRowset,
        L"TABLE_NAME", cchBuffer, pwszBuffer));

CLEANUP:
    if( pIDBSchemaRowset )
        pIDBSchemaRowset->Release();
    if( pUnkRowset )
        pUnkRowset->Release();
    return hr;
}




_
//--------------------------------------------------------------------
// @module ROWSET.CPP
//--------------------------------------------------------------------


////////////////////////////////////////////////////////////////
// Includes
//
////////////////////////////////////////////////////////////////
#include "prsample.h"        // Programmer's Reference Sample includes



////////////////////////////////////////////////////////////////
// myCreateRowset
// Summary of Routines
//
// This function creates an OLE DB rowset object from the given
// provider's session object. It first obtains a default table
// name from the user through the tables schema rowset, if
// supported, and then creates a rowset object by one of two methods:
//
// - If the user requested that the rowset object be created
//     from a command object, it creates a command object and then
//     obtains command text from the user, sets properties and
```

```
//    the command text, and finally executes the command to
//    create the rowset object.
// - Otherwise, the function obtains a table name from the user
//    and calls IOpenRowset::OpenRowset to create a rowset object
//    over that table that supports the requested properties.
//
//////////////////////////////////////////////////////////////
HRESULT     myCreateRowset
    (
    IUnknown *      pUnkSession,
    IUnknown **     ppUnkRowset
    )
{
    HRESULT         hr;
    IUnknown *      pUnkCommand                 = NULL;
    IOpenRowset *   pIOpenRowset                = NULL;
    WCHAR           wszTableName[MAX_NAME_LEN + 1]   = {0};

    const ULONG     cProperties                 = 2;
    DBPROP          rgProperties[cProperties];
    DBPROPSET       rgPropSets[1];

    // Obtain a default table name from the user by displaying the
    // tables schema rowset if schema rowsets are supported.
   CHECK_HR(hr = myCreateSchemaRowset(DBSCHEMA_TABLES,
   pUnkSession,
          MAX_NAME_LEN, wszTableName));

    // Set properties on the rowset, to request additional functionality.
     myAddRowsetProperties(rgPropSets, cProperties, rgProperties);

    // If the user requested that the rowset be created from a
    // Ccommand object, create a command, set its properties and
    // text, and execute it to create the rowset object.
    if( g_dwFlags & USE_COMMAND )
    {
        WCHAR           wszCommandText[MAX_NAME_LEN + 1];

        // Attempt to create the command object from the provider's
        // session object. Note that commands are not supported by
        // all providers, and this will fail in that case.
     CHECK_HR(hr = myCreateCommand(pUnkSession,
     &pUnkCommand));

          // From the user, get the command text that we will execute.
     if( !myGetInputFromUser(wszCommandText, L"\nType the command "
               L"to execute [Enter = `select * from %s`]: ", wszTableName) )
        {
            swprintf(wszCommandText, L"select * from %s", wszTableName);
        }

        // And execute the command the user entered.
     CHECK_HR(hr = myExecuteCommand(pUnkCommand,
     wszCommandText,
                        1, rgPropSets, ppUnkRowset));
    }
    // Otherwise, the user gets the default behavior, which is to use
```

```
    // IOpenRowset to create the rowset object from the session object.
    // IOpenRowset is supported by all providers; it takes a TableID
    // and creates a rowset containing all rows in that table. It is
    // similar to using SQL command text of "SELECT * FROM TableID".
    else
    {
        DBID TableID;

        // Create the TableID.
        TableID.eKind          = DBKIND_NAME;
        TableID.uName.pwszName   = wszTableName;

        // Obtain the table name from the user.
     myGetInputFromUser(wszTableName, L"\nType the name of the table "
                L"to use [Enter = `%s`]: ", wszTableName);

        // Get the IOpenRowset interface, and create a rowset object
        // over the requested table through OpenRowset.

        XCHECK_HR(hr = pUnkSession->QueryInterface(
                    IID_IOpenRowset, (void**)&pIOpenRowset));
        XCHECK_HR(hr = pIOpenRowset->OpenRowset(
                    NULL,                       // pUnkOuter
                    &TableID,                    // pTableID
                    NULL,                       // pIndexID
                    IID_IRowset,                   // riid
                    1,                        // cPropSets
                    rgPropSets,                    // rgPropSets
                    ppUnkRowset                    // ppRowset
                    ));
    }

CLEANUP:
    if( pIOpenRowset )
        pIOpenRowset->Release();
    if( pUnkCommand )
        pUnkCommand->Release();
    return hr;
}


//////////////////////////////////////////////////////////////////////
// mySetupBindings
// Summary of Routines
//
// This function takes an IUnknown pointer from a rowset object
// and creates a bindings array that describes how we want the
// data we fetch from the rowset to be laid out in memory. It
// also calculates the total size of a row so that we can use
// this to allocate memory for the rows that we will fetch
// later.
//
// For each column in the rowset, there will be a corresponding
// element in the bindings array that describes how the
// provider should transfer the data, including length and
// status, for that column. This element also specifies the data
// type that the provider should return the column as. We will
// bind all columns as DBTYPE_WSTR, with a few exceptions
// detailed below, as providers are required to support the
```

```
// conversion of their column data to this type in the vast
// majority of cases. The exception to our binding as
// DBTYPE_WSTR is if the native column data type is
// DBTYPE_IUNKNOWN or if the user has requested that BLOB
// columns be bound as ISequentialStream objects, in which case
// we will bind those columns as ISequentialStream objects.
//
//////////////////////////////////////////////////////////////
HRESULT mySetupBindings
    (
    IUnknown *      pUnkRowset,
    ULONG *         pcBindings,
    DBBINDING **    prgBindings,
    ULONG *         pcbRowSize
    )
{
    HRESULT         hr;
    ULONG           cColumns;
    DBCOLUMNINFO *  rgColumnInfo    = NULL;
    LPWSTR          pStringBuffer   = NULL;
    IColumnsInfo *  pIColumnsInfo   = NULL;

    ULONG           iCol;
    ULONG           dwOffset        = 0;
    DBBINDING *     rgBindings      = NULL;

    ULONG           cStorageObjs    = 0;
    BOOL            fMultipleObjs   = FALSE;

    // Obtain the column information for the rowset; from this, we can
    // find out the following information that we need to construct the
    // bindings array:
    // - the number of columns
    // - the ordinal of each column
    // - the precision and scale of numeric columns
    // - the OLE DB data type of the column
        XCHECK_HR(hr = pUnkRowset->QueryInterface(
                IID_IColumnsInfo, (void**)&pIColumnsInfo));
    XCHECK_HR(hr = pIColumnsInfo->GetColumnInfo(
                &cColumns,                  // pcColumns
                &rgColumnInfo,               // prgColumnInfo
                &pStringBuffer              // ppStringBuffer
                ));

    // Allocate memory for the bindings array; there is a one-to-one
    // mapping between the columns returned from GetColumnInfo and our
    // bindings.
    rgBindings = (DBBINDING*)CoTaskMemAlloc(cColumns * sizeof(DBBINDING));
    CHECK_MEMORY(hr, rgBindings);
    memset(rgBindings, 0, cColumns * sizeof(DBBINDING));

    // Determine if the rowset supports multiple storage object bindings.
    // If it does not, we will bind only the first BLOB column or IUnknown
    // column as an ISequentialStream object, and we will bind the rest as
    // DBTYPE_WSTR.
    myGetProperty(pUnkRowset, IID_IRowset,
    DBPROP_MULTIPLESTORAGEOBJECTS,
        DBPROPSET_ROWSET, &fMultipleObjs);
```

```
    // Construct the binding array element for each column.
    for( iCol = 0; iCol < cColumns; iCol++ )
    {
        // This binding applies to the ordinal of this column.
        rgBindings[iCol].iOrdinal = rgColumnInfo[iCol].iOrdinal;

        // We are asking the provider to give us the data for this column
        // (DBPART_VALUE), the length of that data (DBPART_LENGTH), and
        // the status of the column (DBPART_STATUS).
        rgBindings[iCol].dwPart                                                    =
DBPART_VALUE|DBPART_LENGTH|DBPART_STATUS;

        // The following values are the offsets to the status, length, and
        // data value that the provider will fill with the appropriate
        // values when we fetch data later. When we fetch data, we will
        // pass a pointer to a buffer that the provider will copy column
        // data to, in accordance with the binding we have provided for
        // that column; these are offsets into that future buffer.
        rgBindings[iCol].obStatus   = dwOffset;
        rgBindings[iCol].obLength   = dwOffset + sizeof(DBSTATUS);
        rgBindings[iCol].obValue    = dwOffset + sizeof(DBSTATUS) + sizeof(ULONG);

        // Any memory allocated for the data value will be owned by us, the
        // client. Note that no data will be allocated in this case, as the
        // DBTYPE_WSTR bindings we are using will tell the provider to
        // simply copy data directly into our provided buffer.
        rgBindings[iCol].dwMemOwner     = DBMEMOWNER_CLIENTOWNED;

        // This is not a parameter binding.
        rgBindings[iCol].eParamIO     = DBPARAMIO_NOTPARAM;

        // We want to use the precision and scale of the column.
        rgBindings[iCol].bPrecision   = rgColumnInfo[iCol].bPrecision;
        rgBindings[iCol].bScale       = rgColumnInfo[iCol].bScale;

        // Bind this column as DBTYPE_WSTR, which tells the provider to
        // copy a Unicode string representation of the data into our
        // buffer, converting from the native type if necessary.
        rgBindings[iCol].wType = DBTYPE_WSTR;

        // Initially, we set the length for this data in our buffer to 0;
        // the correct value for this will be calculated directly below.
        rgBindings[iCol].cbMaxLen = 0;

        // Determine the maximum number of bytes required in our buffer to
        // contain the Unicode string representation of the provider's
        // native data type, including room for the NULL-termination
        // character.
        switch( rgColumnInfo[iCol].wType )
        {
            case DBTYPE_NULL:
            case DBTYPE_EMPTY:
            case DBTYPE_I1:
            case DBTYPE_I2:
            case DBTYPE_I4:
            case DBTYPE_UI1:
            case DBTYPE_UI2:
            case DBTYPE_UI4:
            case DBTYPE_R4:
```

```
        case DBTYPE_BOOL:
        case DBTYPE_I8:
        case DBTYPE_UI8:
        case DBTYPE_R8:
        case DBTYPE_CY:
        case DBTYPE_ERROR:
            // When the above types are converted to a string, they
            // will all fit into 25 characters, so use that plus space
            // for the NULL terminator.
            rgBindings[iCol].cbMaxLen = (25 + 1) * sizeof(WCHAR);
            break;

        case DBTYPE_DECIMAL:
        case DBTYPE_NUMERIC:
        case DBTYPE_DATE:
        case DBTYPE_DBDATE:
        case DBTYPE_DBTIMESTAMP:
        case DBTYPE_GUID:
            // Converted to a string, the above types will all fit into
            // 50 characters, so use that plus space for the terminator.
            rgBindings[iCol].cbMaxLen = (50 + 1) * sizeof(WCHAR);
            break;

        case DBTYPE_BYTES:
            // In converting DBTYPE_BYTES to a string, each byte
            // becomes two characters (e.g. 0xFF -> "FF"), so we
            // will use double the maximum size of the column plus
            // include space for the NULL terminator.
            rgBindings[iCol].cbMaxLen =
                (rgColumnInfo[iCol].ulColumnSize * 2 + 1) * sizeof(WCHAR);
            break;

        case DBTYPE_STR:
        case DBTYPE_WSTR:
        case DBTYPE_BSTR:
            // Going from a string to our string representation,
            // we can just take the maximum size of the column,
            // a count of characters, and include space for the
            // terminator, which is not included in the column size.
            rgBindings[iCol].cbMaxLen =
                (rgColumnInfo[iCol].ulColumnSize + 1) * sizeof(WCHAR);
            break;

        default:
            // For any other type, we will simply use our maximum
            // column buffer size, since the display size of these
            // columns may be variable (e.g. DBTYPE_VARIANT) or
            // unknown (e.g. provider-specific types).
            rgBindings[iCol].cbMaxLen = MAX_COL_SIZE;
            break;
    };

    // If the provider's native data type for this column is
    // DBTYPE_IUNKNOWN or this is a BLOB column and the user
    // has requested that we bind BLOB columns as ISequentialStream
    // objects, bind this column as an ISequentialStream object if
    // the provider supports our creating another ISequentialStream
    // binding.
    if( (rgColumnInfo[iCol].wType == DBTYPE_IUNKNOWN ||
```

```
            ((rgColumnInfo[iCol].dwFlags & DBCOLUMNFLAGS_ISLONG) &&
            (g_dwFlags & USE_ISEQSTREAM))) &&
            (fMultipleObjs || !cStorageObjs) )
        {
            // To create an ISequentialStream object, we will
            // bind this column as DBTYPE_IUNKNOWN to indicate
            // that we are requesting this column as an object.
            rgBindings[iCol].wType = DBTYPE_IUNKNOWN;

            // We want to allocate enough space in our buffer for
            // the ISequentialStream pointer we will obtain from
            // the provider.
            rgBindings[iCol].cbMaxLen = sizeof(ISequentialStream *);

            // To specify the type of object that we want from the
            // provider, we need to create a DBOBJECT structure and
            // place it in our binding for this column.
            rgBindings[iCol].pObject =
                                (DBOBJECT *)CoTaskMemAlloc(sizeof(DBOBJECT));
            CHECK_MEMORY(hr, rgBindings[iCol].pObject);


              // Direct the provider to create an ISequentialStream
            // object over the data for this column.
            rgBindings[iCol].pObject->iid = IID_ISequentialStream;

            // We want read access on the ISequentialStream
            // object that the provider will create for us.
            rgBindings[iCol].pObject->dwFlags = STGM_READ;

            // Keep track of the number of storage objects
            // (ISequentialStream is a storage interface) that we have
            // requested, so that we can avoid requesting multiple storage
            // objects from a provider that supports only a single storage
            // object in our bindings.
            cStorageObjs++;
        }

        // Ensure that the bound maximum length is no more than the
        // maximum column size in bytes that we've defined.
        rgBindings[iCol].cbMaxLen
            = min(rgBindings[iCol].cbMaxLen, MAX_COL_SIZE);

        // Update the offset past the end of this column's data so
        // that the next column will begin in the correct place in
        // the buffer.
        dwOffset = rgBindings[iCol].cbMaxLen + rgBindings[iCol].obValue;

        // Ensure that the data for the next column will be correctly
        // aligned for all platforms, or if we're done with columns,
        // that if we allocate space for multiple rows that the data
        // for every row is correctly aligned.
        dwOffset = ROUNDUP(dwOffset);
    }

    // Return the row size (the current dwOffset is the size of the row),
    // the count of bindings, and the bindings array to the caller.
    *pcbRowSize    = dwOffset;
    *pcBindings    = cColumns;
    *prgBindings   = rgBindings;
```

```
CLEANUP:
    CoTaskMemFree(rgColumnInfo);
    CoTaskMemFree(pStringBuffer);
    if( pIColumnsInfo )
        pIColumnsInfo->Release();
    return hr;
}



//////////////////////////////////////////////////////////////
// myCreateAccessor
// Summary of Routines
//
// This function takes an IUnknown pointer for a rowset object
// and creates an accessor that describes the layout of the
// buffer we will use when we fetch data. The provider will fill
// this buffer according to the description contained in the
// accessor that we will create here.
//
//////////////////////////////////////////////////////////////
HRESULT myCreateAccessor
    (
    IUnknown *     pUnkRowset,
    HACCESSOR *    phAccessor,
    ULONG *        pcBindings,
    DBBINDING **   prgBindings,
    ULONG *        pcbRowSize
    )
{
    HRESULT        hr;
    IAccessor *    pIAccessor = NULL;

    // An accessor is basically a handle to a collection of bindings.
    // To create the accessor, we need to first create an array of
    // bindings for the columns in the rowset.
CHECK_HR(hr = mySetupBindings(pUnkRowset, pcBindings, prgBindings,
        pcbRowSize));

    // Now that we have an array of bindings, tell the provider to
    // create the accessor for those bindings. We get back a handle
    // to this accessor, which we will use when fetching data.
      XCHECK_HR(hr = pUnkRowset->QueryInterface(
                IID_IAccessor, (void**)&pIAccessor));
    XCHECK_HR(hr = pIAccessor->CreateAccessor(
                DBACCESSOR_ROWDATA,              // dwAccessorFlags
                *pcBindings,                     // cBindings
                *prgBindings,                    // rgBindings
                0,                               // cbRowSize
                phAccessor,                      // phAccessor
                NULL                             // rgStatus
                ));

CLEANUP:
    if( pIAccessor )
        pIAccessor->Release();
    return hr;
}
```

```
//////////////////////////////////////////////////////////////
// myDisplayRowset
// Summary of Routines
//
// This function will display data from a rowset object and will
// allow the user to perform basic navigation of the rowset.
//
// The function takes a pointer to a rowset object's IUnknown
// and, optionally, the name of a column and a buffer that will
// receive the value of that column when the user selects a row.
//
//////////////////////////////////////////////////////////////
HRESULT myDisplayRowset
    (
    IUnknown *    pUnkRowset,
    LPCWSTR       pwszColToReturn,
    ULONG         cchBuffer,
    LPWSTR        pwszBuffer
    )
{
    HRESULT       hr;
    IRowset *     pIRowset              = NULL;
    ULONG         cBindings;
    DBBINDING *   rgBindings            = NULL;
    HACCESSOR     hAccessor             = DB_NULL_HACCESSOR;
    ULONG         cbRowSize;
    void *        pData                 = NULL;
    ULONG *       rgDispSize            = NULL;
    DBCOUNTITEM   cRowsObtained;
    HROW *        rghRows               = NULL;
    ULONG         iRow;
    LONG          cRows                 = MAX_ROWS;
    LONG          iRetCol               = -1;
    BOOL          fCanFetchBackwards;
    ULONG         iIndex;
    void *        pCurData;

    // Obtain the IRowset interface for use in fetching rows and data
        XCHECK_HR(hr = pUnkRowset->QueryInterface(
                IID_IRowset, (void**)&pIRowset));

    // Determine whether this rowset supports fetching data backwards;
    // we use this to determine whether the rowset can support moving
    // to the previous set of rows, described in more detail below
    myGetProperty(pUnkRowset, IID_IRowset,
    DBPROP_CANFETCHBACKWARDS,
          DBPROPSET_ROWSET, &fCanFetchBackwards);

    // If the caller wants us to return the data for a particular column
    // from a user-selected row, we need to turn the column name into a
    // column ordinal.
    if( pwszColToReturn )
    CHECK_HR(hr = myFindColumn(pUnkRowset, pwszColToReturn,
    &iRetCol));

    // Create an accessor. An accessor is basically a handle to a
```

```
// collection of bindings that describes to the provider how to
// copy (and convert, if necessary) column data into our buffer.
// The accessor that this creates will bind all columns either as
// DBTYPE_WSTR (a Unicode string) or as an ISequentialStream object
// (used for BLOB data). This will also give us the size of the
// row buffer that the accessor describes to the provider.
CHECK_HR(hr = myCreateAccessor(pUnkRowset, &hAccessor,
                               &cBindings, &rgBindings, &cbRowSize));

// Allocate enough memory to hold cRows rows of data; this is
// where the actual row data from the provider will be placed.
pData = CoTaskMemAlloc(cbRowSize * MAX_ROWS);
CHECK_MEMORY(hr, pData);

// Allocate memory for an array that we will use to calculate the
// maximum display size used by each column in the current set of
// rows.
rgDispSize = (ULONG *)CoTaskMemAlloc(cBindings * sizeof(ULONG));
CHECK_MEMORY(hr, rgDispSize);

// In this loop, we perform the following process:
// - reset the maximum display size array
// - try to get cRows row handles from the provider
// - these handles are then used to actually get the row data from the
//    provider copied into our allocated buffer
// - calculate the maximum display size for each column
// - release the row handles to the rows we obtained
// - display the column names for the rowset
// - display the row data for the rows that we fetched
// - get user input
// - free the provider-allocated row handle array
// - repeat unless the user has chosen to quit or has selected a row
while( hr == S_OK )
{
    // Clear the maximum display size array.
    memset(rgDispSize, 0, cBindings * sizeof(ULONG));

    // Attempt to get cRows row handles from the provider.
    XCHECK_HR(hr = pIRowset->GetNextRows(
                DB_NULL_HCHAPTER,                   // hChapter
                0,                              // lOffset
                cRows,                          // cRows
                &cRowsObtained,                     // pcRowsObtained
                &rghRows                        // prghRows
                ));

    // Loop over the row handles obtained from GetNextRows,
    // actually fetching the data for these rows into our buffer.
    for( iRow = 0; iRow < cRowsObtained; iRow++ )
    {
        // Find the location in our buffer where we want to place
        // the data for this row. Note that if we fetched rows
        // backwards (cRows < 0), the row handles obtained from the
        // provider are reversed from the order in which we want to
        // actually display the data on the screen, so we will
        // account for this. This ensures that the resulting order
        // of row data in the pData buffer matches the order we
        // wish to use to display the data.
        iIndex     = cRows > 0 ? iRow : cRowsObtained - iRow - 1;
```

```
    pCurData   = (BYTE*)pData + (cbRowSize * iIndex);

    // Get the data for this row handle. The provider will copy
    // (and convert, if necessary) the data for each of the
    // columns that are described in our Aaccessor into the given
    // buffer (pCurData).
    XCHECK_HR(hr = pIRowset->GetData(
            rghRows[iRow],                  // hRow
            hAccessor,                      // hAccessor
            pCurData                        // pData
            ));

    // Update the maximum display size array, accounting for
    // this row.
   CHECK_HR(hr = myUpdateDisplaySize(cBindings, rgBindings,
          pCurData, rgDispSize));
}

// If we obtained rows, release the row handles for the retrieved
// rows and display the names of the rowset columns before we
// display the data.
if( cRowsObtained )
{
    // Release the row handles that we obtained.
    XCHECK_HR(hr = pIRowset->ReleaseRows(
                cRowsObtained,                  // cRows
                rghRows,                        // rghRows
                NULL,                           // rgRowOptions
                NULL,                           // rgRefCounts
                NULL                            // rgRowStatus
                ));


    // Display the names of the rowset columns.
     CHECK_HR(hr = myDisplayColumnNames(pIRowset,
     rgDispSize));
 }

// For each row that we obtained the data for, display this data.
for( iRow = 0; iRow < cRowsObtained; iRow++ )
{
    // Get a pointer to the data for this row.
    pCurData = (BYTE*)pData + (cbRowSize* iRow);

    // And display the row data.
     CHECK_HR(hr = myDisplayRow(iRow, cBindings, rgBindings,
          pCurData, rgDispSize));
}

// Allow the user to navigate the rowset. This displays the
// appropriate prompts, gets the user's input, may call
// IRowset::RestartPosition, and may copy data from a selected row
// to the selection buffer, if so directed. This will return S_OK
// if the user asked for more rows, S_FALSE if the user selected a
// row, or E_FAIL if the user quits.
hr = myInteractWithRowset(
     pIRowset,              // IRowset pointer, for RestartPosition
    &cRows,                 // updated with fetch direction value
```

```
            cRowsObtained,          // to indicate selection range
            fCanFetchBackwards,     // whether [P]revious is supported
            pData,                  // data pointer for copying selection
            cbRowSize,              // size of rows for copying selection
            iRetCol >= 0 ?          // bindings for the selection column,
                &rgBindings[iRetCol] :   // or NULL if no selection column
                NULL,
            cchBuffer,              // size of the selection buffer
            pwszBuffer);            // pointer to the selection buffer

        // Since we are allowing the provider to allocate the memory for
        // the row handle array, we will free this memory and reset the
        // pointer to NULL. If this is not NULL on the next call to
        // GetNextRows, the provider will assume that it points to an
        // allocated array of the required size (which may not be the case
        // if we obtained less than cRows rows from this last call to
        // GetNextRows).
        CoTaskMemFree(rghRows);
        rghRows = NULL;
    }

CLEANUP:
myFreeBindings(cBindings, rgBindings);
     CoTaskMemFree(rgDispSize);
    CoTaskMemFree(pData);
    if( pIRowset )
        pIRowset->Release();
    return hr;
}



//////////////////////////////////////////////////////////////////
// myInteractWithRowset
// Summary of Routines
//
// This function allows the user to interact with the rowset. It
// prompts the user appropriately, gets the user's input, may
// call IRowset::RestartPosition if the user requests a restart,
// and will copy data from a selected row to the selection
// buffer.
//
//////////////////////////////////////////////////////////////////
HRESULT myInteractWithRowset
    (
    IRowset *     pIRowset,
    LONG *        pcRows,
    DBCOUNTITEM   cRowsObtained,
    BOOL          fCanFetchBackwards,
    void *        pData,
    ULONG         cbRowSize,
    DBBINDING *   pBinding,
    ULONG         cchBuffer,
    LPWSTR        pwszBuffer
    )
{
    HRESULT       hr = S_OK;
    CHAR          ch;
```

```
// Let the user know if no rows were fetched.
if( !cRowsObtained )
    printf("\n*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*\n" \
           "*                                        *\n" \
           "* No rows obtained on this fetch! *\n" \
           "*                                        *\n" \
           "*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*\n");

// Print navigation options.
if( fCanFetchBackwards )
    printf("\n[P]revious; [N]ext; [R]estart; ");
else
    printf("\n[N]ext; [R]estart; ");

// Print selection options.
if( cRowsObtained && pwszBuffer && pBinding )
    printf("[0]-[%d] for a row; ", cRowsObtained - 1);

// User can always quit the program.
printf("[Q]uit? ");

// Get the user's input.
while( TRUE )
{
    // Get a character from the console.
ch = myGetChar();

    // Look for one of the allowed options; if not found, go
    // back around and wait for another input from the user.

    // If we're looking for a row selection, allow the user to select
    // a row that we fetched, and then copy the data from the requested
    // column into the selection buffer we were passed.
    if( pwszBuffer && pBinding &&
        ch >= '0' && ch < (int)('0' + cRowsObtained) )
    {
        // Save the data for the selected row.
        ULONG nSelection = ch - '0';
        _snwprintf(pwszBuffer, cchBuffer, L"%s",
                   (WCHAR *)((BYTE *)pData + cbRowSize * nSelection +
                   pBinding->obValue));
        pwszBuffer[cchBuffer] = L'\0';
        hr = S_FALSE;
    }
    // If the provider supports fetching backwards, set *pcRows
    // to -MAX_ROWS. When GetNextRows is called with this value,
    // it will fetch rows backwards from the current position
    // until it fetches MAX_ROWS rows or hits the end of the rowset.
    else if( fCanFetchBackwards && ch == 'p' )
    {
        // Fetch backwards.
        *pcRows = -MAX_ROWS;
    }
    // Set *pcRows so that the next call to GetNextRows fetches
    // MAX_ROWS rows forward from the current position.
    else if( ch == 'n' )
    {
        // Fetch forward
        *pcRows = MAX_ROWS;
```

```
        }
        // Call IRowset::RestartPosition, and fetch the first MAX_ROWS
        // rows of the rowset forward from there.
        else if( ch == 'r' )
        {
            // RestartPosition
            *pcRows = MAX_ROWS;
            XCHECK_HR(hr = pIRowset->RestartPosition(DB_NULL_HCHAPTER));

            // Restarting a command may return the DB_S_COMMANDREEXECUTED
            // warning. If this happens, we still want the caller to
            // continue to display data, so we will reset the result code.
            // to S_OK.
            hr = S_OK;
        }
        // Quit the program.
        else if( ch == 'q' )
        {
            hr = E_FAIL;
        }
        // Invalid option; go back up and get another character from the
        // user.
        else
        {
            continue;
        }

        // Echo the character and stop waiting for input.
        printf("%c\n", ch);
        break;
    }

CLEANUP:
    return hr;
}


//////////////////////////////////////////////////////////////////
// myDisplayColumnNames
// Summary of Routines
//
// This function takes an IUnknown pointer to a rowset object
// and displays the names of the columns of that rowset.
//
//////////////////////////////////////////////////////////////////
HRESULT myDisplayColumnNames
    (
    IUnknown *      pUnkRowset,
    ULONG *         rgDispSize
    )
{
    HRESULT         hr;
    IColumnsInfo *  pIColumnsInfo    = NULL;
    ULONG           cColumns;
    DBCOLUMNINFO *  rgColumnInfo     = NULL;
    LPOLESTR        pStringsBuffer   = NULL;
    WCHAR           wszColumn[MAX_DISPLAY_SIZE + 1];
    LPWSTR          pwszColName;
    ULONG           iCol;
```

```
    ULONG          cSpaces;
    ULONG          iSpace;

    // Get the IColumnsInfo interface for the rowset.
  XCHECK_HR(hr = pUnkRowset->QueryInterface(
                IID_IColumnsInfo, (void**)&pIColumnsInfo));

    // Get the columns information.
    XCHECK_HR(hr = pIColumnsInfo->GetColumnInfo(
                &cColumns,                    // pcColumns
                &rgColumnInfo,                 // prgColumnInfo
                &pStringsBuffer                // ppStringBuffer
                ));

    // Display the title of the row index column.
    wprintf(L" Row | ");

    // Display all column names.
    for( iCol = 0; iCol < cColumns; iCol++ )
    {
        pwszColName = rgColumnInfo[iCol].pwszName;

        // If the column name is NULL, we'll use a default string.
        if( !pwszColName )
        {
            // Is this the bookmark column?
            if( !rgColumnInfo[iCol].iOrdinal )
                pwszColName = L"Bmk";
            else
                pwszColName = L"(null)";
        }

        // Ensure that the name is no longer than MAX_DISPLAY_SIZE.
        wcsncpy(wszColumn, pwszColName, MAX_DISPLAY_SIZE);
        wszColumn[min(rgDispSize[iCol], MAX_DISPLAY_SIZE)] = L'\0';

        // Figure out how many spaces we need to print after
        // this column name.
        cSpaces = min(rgDispSize[iCol], MAX_DISPLAY_SIZE) - wcslen(wszColumn);

        // Print the column name.
        wprintf(L"%s", wszColumn);

        // Now print any spaces necessary to align this column.
        for(iSpace = 0; iSpace < cSpaces; iSpace++ )
            putch(' ');

        // Now end the column with a separator marker if necessary.
        if( iCol < cColumns - 1 )
            wprintf(L" | ");
    }

    // Done with the header, so print a new line.
    wprintf(L"\n");

CLEANUP:
    CoTaskMemFree(rgColumnInfo);
    CoTaskMemFree(pStringsBuffer);
    if( pIColumnsInfo )
```

```
        pIColumnsInfo->Release();
    return hr;
}


//////////////////////////////////////////////////////////////
// myDisplayRow
// Summary of Routines
//
// This function displays the data for a row.
//
//////////////////////////////////////////////////////////////
HRESULT myDisplayRow
    (
    ULONG               iRow,
    ULONG               cBindings,
    DBBINDING *           rgBindings,
    void *              pData,
    ULONG *             rgDispSize
    )
{
    HRESULT              hr = S_OK;
    WCHAR               wszColumn[MAX_DISPLAY_SIZE + 1];
    DBSTATUS             dwStatus;
    ULONG               ulLength;
    void *              pvValue;
    ULONG               iCol;
    ULONG               cbRead;
    ISequentialStream *   pISeqStream = NULL;
    ULONG               cSpaces;
    ULONG               iSpace;

    // Display the row number.
    wprintf(L" [%d] | ", iRow);

    // For each column that we have bound, display the data.
    for( iCol = 0; iCol < cBindings; iCol++ )
    {
        // We have bound status, length, and the data value for all
        // columns, so we know that these can all be used.
        dwStatus   = *(DBSTATUS *)((BYTE *)pData + rgBindings[iCol].obStatus);
        ulLength   = *(ULONG *)((BYTE *)pData + rgBindings[iCol].obLength);
        pvValue    = (BYTE *)pData + rgBindings[iCol].obValue;

        // Check the status of this column. This decides
        // exactly what will be displayed for the column.
        switch( dwStatus )
        {
            // The data is NULL, so don't try to display it.
            case DBSTATUS_S_ISNULL:
                wcscpy(wszColumn, L"(null)");
                break;

            // The data was fetched, but may have been truncated.
            // Display string data for this column to the user.
            case DBSTATUS_S_TRUNCATED:
            case DBSTATUS_S_OK:
            case DBSTATUS_S_DEFAULT:
            {
```

```
            // We have bound the column either as a Unicode string
            // (DBTYPE_WSTR) or as an ISequentialStream object
            // (DBTYPE_IUNKNOWN), and we have to do different processing
            // for each one of these possibilities.
            switch( rgBindings[iCol].wType )
            {
                case DBTYPE_WSTR:
                {
                    // Copy the string data.
                    wcsncpy(wszColumn, (WCHAR *)pvValue, MAX_DISPLAY_SIZE);
                    wszColumn[MAX_DISPLAY_SIZE - 1] = L'\0';
                    break;
                }

                case DBTYPE_IUNKNOWN:
                {
                    // We've bound this as an ISequentialStream object,
                    // therefore the data in our buffer is a pointer
                    // to the object's ISequentialStream interface.
                    pISeqStream = *(ISequentialStream**)pvValue;

                    // We call ISequentialStream::Read to read bytes from
                    // the stream blindly into our buffer, simply as a
                    // demonstration of ISequentialStream. To display the
                    // data properly, the native provider type of this
                    // column should be accounted for; it could be
                    // DBTYPE_WSTR, in which case this works, or it could
                    // be DBTYPE_STR or DBTYPE_BYTES, in which case this
                    // won't display the data correctly.
                    CHECK_HR(hr = pISeqStream->Read(
                                wszColumn,              // pBuffer
                                MAX_DISPLAY_SIZE,       // cBytes
                                &cbRead                 // pcBytesRead
                                ));

                    // Since streams don't provide NULL-termination,
                    // we'll NULL-terminate the resulting string ourselves.
                    wszColumn[cbRead / sizeof(WCHAR)] = L'\0';

                    // Release the stream object, now that we're done.
                    pISeqStream->Release();
                    pISeqStream = NULL;
                    break;
                }
            }
            break;
        }

        // This is an error status, so don't try to display the data.
        default:
            wcscpy(wszColumn, L"(error status)");
            break;
    }

    // Determine how many spaces we need to add after displaying this
    // data to align it with this column in other rows.
    cSpaces = min(rgDispSize[iCol], MAX_DISPLAY_SIZE) - wcslen(wszColumn);

    // Print the column data.
```

```
            wprintf(L"%s", wszColumn);

            // Now print any spaces necessary.
            for(iSpace = 0; iSpace < cSpaces; iSpace++ )
                putch(' ');

            // Now end the column with a separator marker if necessary.
            if( iCol < cBindings - 1 )
                wprintf(L" | ");
        }

    CLEANUP:
        if( pISeqStream )
            pISeqStream->Release();

        // Print the row separator.
        wprintf(L"\n");
        return hr;
    }



    //////////////////////////////////////////////////////////////
    // myFreeBindings
    // Summary of Routines
    //
    //     This function frees a bindings array and any allocated
    // structures contained in that array.
    //
    //////////////////////////////////////////////////////////////
    void myFreeBindings
        (
        ULONG        cBindings,
        DBBINDING *   rgBindings
        )
    {
        ULONG        iBind;

        // Free any memory used by DBOBJECT structures in the array.
        for( iBind = 0; iBind < cBindings; iBind++ )
            CoTaskMemFree(rgBindings[iBind].pObject);

        // Now free the bindings array itself.
        CoTaskMemFree(rgBindings);
    }


    //////////////////////////////////////////////////////////////
    // myAddRowsetProperties
    // Summary of Routines
    //
    // This function sets up the given DBPROPSET and DBPROP
    // structures, adding two optional properties that describe
    // features that we would like to use on the rowset created
    // with these properties applied:
    // - DBPROP_CANFETCHBACKWARDS -- the rowset should support
    //    fetching rows backwards from our current cursor position.
    // - DBPROP_IRowsetLocate -- the rowset should support
    //    the IRowsetLocate interface and its semantics.
```

```
//
//////////////////////////////////////////////////////////////
void myAddRowsetProperties(DBPROPSET* pPropSet, ULONG cProperties, DBPROP*
rgProperties)
{
    // Initialize the property set array.
    pPropSet->rgProperties      = rgProperties;
    pPropSet->cProperties       = cProperties;
    pPropSet->guidPropertySet   = DBPROPSET_ROWSET;

    // Add the following two properties (as OPTIONAL) to the property
    // array contained in the property set array in order to request
    // that they be supported by the rowset we will create. Because
    // these are optional, the rowset we obtain may or may not support
    // this functionality. We will check for the functionality that
    // we need once the rowset is created and will modify our behavior
    // appropriately.
  myAddProperty(&rgProperties[0], DBPROP_CANFETCHBACKWARDS);
  myAddProperty(&rgProperties[1], DBPROP_IRowsetLocate);
}


//////////////////////////////////////////////////////////////
// myUpdateDisplaySize
// Summary of Routines
//
// This function updates the rgDispSize array, keeping the
// maximum of the display size needed for the given data and
// the previous maximum size already in the array.
//
//////////////////////////////////////////////////////////////
HRESULT myUpdateDisplaySize
    (
    ULONG        cBindings,
    DBBINDING *   rgBindings,
    void *       pData,
    ULONG *      rgDispSize
    )
{
    DBSTATUS     dwStatus;
    ULONG        cchLength;
    ULONG        iCol;

    // Loop through the bindings, comparing the size of each column
    // against the previously found maximum size for that column.
    for( iCol = 0; iCol < cBindings; iCol++ )
    {
        dwStatus = *(DBSTATUS *)((BYTE *)pData + rgBindings[iCol].obStatus);
        cchLength = ((*(ULONG *)((BYTE *)pData + rgBindings[iCol].obLength))
                    / sizeof(WCHAR));

        // The length that we need to display depends on the status
        // of this column and generally on the data in the column.
        switch( dwStatus )
        {
            case DBSTATUS_S_ISNULL:
                cchLength = 6;                      // "(null)"
                break;
```

```
            case DBSTATUS_S_TRUNCATED:
            case DBSTATUS_S_OK:
            case DBSTATUS_S_DEFAULT:
                if( rgBindings[iCol].wType == DBTYPE_IUNKNOWN )
                    cchLength = 2 + 8;                  // "0x%08lx"

                // Ensure that the length is at least the minimum
                // display size.
                cchLength = max(cchLength, MIN_DISPLAY_SIZE);
                break;

            default:
                cchLength = 14;                     // "(error status)"
                break;
        }

        if( rgDispSize[iCol] < cchLength )
            rgDispSize[iCol] = cchLength;
    }

    return S_OK;
}


//////////////////////////////////////////////////////////////////
// myFindColumn
// Summary of Routines
//
// Find the index of the column described in pwszName, and return
// S_OK or, if not found, S_FALSE.
//
//////////////////////////////////////////////////////////////////
HRESULT myFindColumn
    (
    IUnknown *      pUnkRowset,
    LPCWSTR         pwszName,
    LONG *          plIndex
    )
{
    HRESULT         hr;
    IColumnsInfo *  pIColumnsInfo    = NULL;
    ULONG           cColumns;
    DBCOLUMNINFO *  rgColumnInfo     = NULL;
    OLECHAR *       pStringsBuffer   = NULL;
    ULONG           iCol;

    // Get the IColumnsInfo interface.
    XCHECK_HR(hr = pUnkRowset->QueryInterface(
                IID_IColumnsInfo, (void**)&pIColumnsInfo));

    // Get the columns information.
    XCHECK_HR(hr = pIColumnsInfo->GetColumnInfo(
                &cColumns,                      // pcColumns
                &rgColumnInfo,                  // prgColumnInfo
                &pStringsBuffer                 // ppStringBuffer
                ));

    // Assume that we'll find the column.
    hr = S_OK;
```

```
    // Search for the column we need.
    for( iCol = 0; iCol < cColumns; iCol++ )
    {
        // If the column name matches, we've found the column....
        if( rgColumnInfo[iCol].pwszName &&
            !wcscmp(pwszName, rgColumnInfo[iCol].pwszName) )
        {
            *plIndex = iCol;
            goto CLEANUP;
        }
    }

    // If we didn't find the column, we'll return S_FALSE.
    hr = S_FALSE;

CLEANUP:
    CoTaskMemFree(rgColumnInfo);
    CoTaskMemFree(pStringsBuffer);
    if( pIColumnsInfo )
        pIColumnsInfo->Release();
    return hr;
}



//--------------------------------------------------------------------
// @module COMMAND.CPP
//--------------------------------------------------------------------


////////////////////////////////////////////////////////////////////
// Includes
//
////////////////////////////////////////////////////////////////////
#include "prsample.h"        // Programmer's Reference Sample includes


////////////////////////////////////////////////////////////////////
// myCreateCommand
// Summary of Routines
//
// This function takes an IUnknown pointer on a session object
// and attempts to create a command object using the session's
// IDBCreateCommand interface. Since this interface is optional,
// this may fail.
//
////////////////////////////////////////////////////////////////////
HRESULT myCreateCommand
    (
    IUnknown *          pUnkSession,
    IUnknown **         ppUnkCommand
    )
{
    HRESULT             hr;
    IDBCreateCommand *  pIDBCreateCommand = NULL;
```

```
    // Attempt to create a command object from the session object
    XCHECK_HR(hr = pUnkSession->QueryInterface(
                IID_IDBCreateCommand, (void**)&pIDBCreateCommand));
    XCHECK_HR(hr = pIDBCreateCommand->CreateCommand(
                NULL,                           // pUnkOuter
                IID_ICommand,                   // riid
                ppUnkCommand                    // ppCommand
                ));

CLEANUP:
    if( pIDBCreateCommand )
        pIDBCreateCommand->Release();
    return hr;
}



//////////////////////////////////////////////////////////////
// myExecuteCommand
// Summary of Routines
//
// This function takes an IUnknown pointer on a command object
// and performs the following steps to create a new rowset
// object:
// - sets the given properties on the command object; these
//    properties will be applied by the provider to any rowset
//    created by this command.
// - sets the given command text for the command.
// - executes the command to create a new rowset object.
//
//////////////////////////////////////////////////////////////
HRESULT myExecuteCommand
    (
    IUnknown *          pUnkCommand,
    WCHAR *             pwszCommandText,
    ULONG               cPropSets,
    DBPROPSET*          rgPropSets,
    IUnknown **         ppUnkRowset
    )
{
    HRESULT             hr;
    ICommandText *          pICommandText       = NULL;
    ICommandProperties *    pICommandProperties   = NULL;

    // Set the properties on the command object.
   XCHECK_HR(hr = pUnkCommand->QueryInterface(
                IID_ICommandProperties, (void**)&pICommandProperties));
    XCHECK_HR(hr = pICommandProperties->SetProperties(cPropSets, rgPropSets));

    // Set the text for this command, using the default command text
    // dialect. All providers that support commands must support this
    // dialect, and providers that support SQL must be able to recognize
    // an SQL command as SQL when this dialect is specified.
   XCHECK_HR(hr = pUnkCommand->QueryInterface(
                IID_ICommandText, (void**)&pICommandText));
    XCHECK_HR(hr = pICommandText->SetCommandText(
                DBGUID_DEFAULT,                 // guidDialect
                pwszCommandText                 // pwszCommandText
```

```
                ));

    // And execute the command. Note that the user could have
    // entered a non-row returning command, so we will check for
    // that and return failure to prevent the display of the
    // nonexistent rowset by the caller.
    XCHECK_HR(hr = pICommandText->Execute(
                NULL,                       // pUnkOuter
                IID_IRowset,                  // riid
                NULL,                       // pParams
                NULL,                       // pcRowsAffected
                ppUnkRowset                     // ppRowset
                ));

    if( !*ppUnkRowset )
    {
        printf("\nThe command executed successfully, but did not " \
                 "return a rowset.\nNo rowset will be displayed.\n");
        hr = E_FAIL;
    }


CLEANUP:
    if( pICommandText )
        pICommandText->Release();
    if( pICommandProperties )
        pICommandProperties->Release();
    return hr;
}




//--------------------------------------------------------------------
// @module ERROR.CPP
//--------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////
// Includes
//
////////////////////////////////////////////////////////////////////////
#include "prsample.h"        // Programmer's Reference Sample includes


////////////////////////////////////////////////////////////////////////
// myHandleResult
// Summary of Routines
//
// This function is called as part of the XCHECK_HR macro; it takes an
// HRESULT, which is returned by the method called in the XCHECK_HR
// macro, and the file and line number where the method call was made.
// If the method call failed, this function attempts to get and display
// the extended error information for the call from the IErrorInfo,
// IErrorRecords, and ISQLErrorInfo interfaces.
//
////////////////////////////////////////////////////////////////////////
HRESULT myHandleResult
    (
```

```
    HRESULT         hrReturned,
    LPCWSTR         pwszFile,
    ULONG           ulLine
    )
{
    HRESULT         hr;
    IErrorInfo *    pIErrorInfo     = NULL;
    IErrorRecords * pIErrorRecords  = NULL;
    ULONG           cRecords;
    ULONG           iErr;

    // If the method called as part of the XCHECK_HR macro failed,
    // we will attempt to get extended error information for the call.
    if( FAILED(hrReturned) )
    {
        // Obtain the current error object, if any, by using the
        // Automation GetErrorInfo function, which will give
        // us back an IErrorInfo interface pointer if successful.
        hr = GetErrorInfo(0, &pIErrorInfo);

        // We've got the IErrorInfo interface pointer on the error object.
        if( SUCCEEDED(hr) && pIErrorInfo )
        {
            // OLE DB extends the Automation error model by allowing
            // error objects to support the IErrorRecords interface. This
            // interface can expose information on multiple errors.
                hr = pIErrorInfo->QueryInterface(IID_IErrorRecords,
                        (void**)&pIErrorRecords);
            if( SUCCEEDED(hr) )
            {
                // Get the count of error records from the object.
                CHECK_HR(hr = pIErrorRecords->GetRecordCount(&cRecords));

                // Loop through the set of error records, and
                // display the error information for each one.
                for( iErr = 0; iErr < cRecords; iErr++ )
                {
                  myDisplayErrorRecord(hrReturned, iErr, pIErrorRecords,
                            pwszFile, ulLine);
                }
            }
            // The object didn't support IErrorRecords; display
            // the error information for this single error.
            else
            {
             myDisplayErrorInfo(hrReturned, pIErrorInfo, pwszFile, ulLine);
              }
        }
        // There was no error object, so just display the HRESULT
        // to the user.
        else
        {
            wprintf(L"\nNo Error Info posted; HResult: 0x%08x\n"
                L"File: %s, Line: %d\n", hrReturned, pwszFile, ulLine);
        }
    }

CLEANUP:
    if( pIErrorInfo )
```

```
        pIErrorInfo->Release();
    if( pIErrorRecords )
        pIErrorRecords->Release();
    return hrReturned;
}


//////////////////////////////////////////////////////////////////////
// myDisplayErrorRecord
// Summary of Routines
//
// This function displays the error information for a single error
// record, including information from ISQLErrorInfo, if supported.
//
//////////////////////////////////////////////////////////////////////
HRESULT myDisplayErrorRecord
    (
    HRESULT          hrReturned,
    ULONG            iRecord,
    IErrorRecords *  pIErrorRecords,
    LPCWSTR          pwszFile,
    ULONG            ulLine
    )
{
    HRESULT          hr;
    IErrorInfo *     pIErrorInfo      = NULL;
    BSTR             bstrDescription  = NULL;
    BSTR             bstrSource       = NULL;
    BSTR             bstrSQLInfo      = NULL;

    static LCID      lcid             = GetUserDefaultLCID();

    LONG             lNativeError     = 0;
    ERRORINFO        ErrorInfo;

    // Get the IErrorInfo interface pointer for this error record.
    CHECK_HR(hr = pIErrorRecords->GetErrorInfo(iRecord, lcid, &pIErrorInfo));

    // Get the description of this error.
    CHECK_HR(hr = pIErrorInfo->GetDescription(&bstrDescription));

    // Get the source of this error.
    CHECK_HR(hr = pIErrorInfo->GetSource(&bstrSource));

    // Get the basic error information for this record.
    CHECK_HR(hr = pIErrorRecords->GetBasicErrorInfo(iRecord, &ErrorInfo));

    // If the error object supports ISQLErrorInfo, get this information.
    myGetSqlErrorInfo(iRecord, pIErrorRecords, &bstrSQLInfo,
    &lNativeError);

    // Display the error information to the user.
    if( bstrSQLInfo )
    {
        wprintf(L"\nErrorRecord:    HResult: 0x%08x\nDescription: %s\n"
            L"SQLErrorInfo: %s\nSource: %s\nFile: %s, Line: %d\n",
            ErrorInfo.hrError,
            bstrDescription,
            bstrSQLInfo,
```

```
            bstrSource,
            pwszFile,
            ulLine);
    }
    else
    {
        wprintf(L"\nErrorRecord:    HResult: 0x%08x\nDescription: %s\n"
            L"Source: %s\nFile: %s, Line: %d\n",
            ErrorInfo.hrError,
            bstrDescription,
            bstrSource,
            pwszFile,
            ulLine);
    }

CLEANUP:
    if( pIErrorInfo )
        pIErrorInfo->Release();
    SysFreeString(bstrDescription);
    SysFreeString(bstrSource);
    SysFreeString(bstrSQLInfo);
    return hr;
}


//////////////////////////////////////////////////////////////////////
// myDisplayErrorInfo
// Summary of Routines
//
// This function displays basic error information for an error object
// that doesn't support the IErrorRecords interface.
//
//////////////////////////////////////////////////////////////////////
HRESULT myDisplayErrorInfo
    (
    HRESULT       hrReturned,
    IErrorInfo *  pIErrorInfo,
    LPCWSTR       pwszFile,
    ULONG         ulLine
    )
{
    HRESULT       hr;
    BSTR          bstrDescription   = NULL;
    BSTR          bstrSource        = NULL;

    // Get the description of the error.
    CHECK_HR(hr = pIErrorInfo->GetDescription(&bstrDescription));

    // Get the source of the error -- this will be the window title.
    CHECK_HR(hr = pIErrorInfo->GetSource(&bstrSource));

    // Display this error information.
    wprintf(L"\nErrorInfo:    HResult: 0x%08x, Description: %s\nSource:
                %s\n" L"File: %s, Line: %d\n",
                hrReturned,
                bstrDescription,
                bstrSource,
                pwszFile,
                ulLine);
```

```
CLEANUP:
    SysFreeString(bstrDescription);
    SysFreeString(bstrSource);
    return hr;
}


//////////////////////////////////////////////////////////////////////
// myGetSqlErrorInfo
// Summary of Routines
//
// If the error object supports ISQLErrorInfo, get the SQL error
// string and native error code for this error.
//
//////////////////////////////////////////////////////////////////////
HRESULT myGetSqlErrorInfo
    (
    ULONG            iRecord,
    IErrorRecords *   pIErrorRecords,
    BSTR *           pBstr,
    LONG *           plNativeError
    )
{
    HRESULT          hr;
    ISQLErrorInfo *   pISQLErrorInfo   = NULL;
    LONG             lNativeError     = 0;

    // Attempt to get the ISQLErrorInfo interface for this error
    // record through GetCustomErrorObject. Note that ISQLErrorInfo
    // is not mandatory, so failure is acceptable here.
   CHECK_HR(hr = pIErrorRecords->GetCustomErrorObject(
                iRecord,                        // iRecord
                IID_ISQLErrorInfo,                  // riid
                (IUnknown**)&pISQLErrorInfo          // ppISQLErrorInfo
                ));

    // If we obtained the ISQLErrorInfo interface, get the SQL
    // error string and native error code for this error.
    if( pISQLErrorInfo )
        hr = pISQLErrorInfo->GetSQLInfo(pBstr, &lNativeError);

CLEANUP:
    if( plNativeError )
        *plNativeError = lNativeError;
    if( pISQLErrorInfo )
        pISQLErrorInfo->Release();
    return hr;
}
```

4.2 ADO Code Examples in Microsoft Visual Basic

Use the following code examples to learn how to use the ADO methods, properties, and events when writing in Visual Basic.

NOTE: Paste the entire code example, from Sub to End Sub, into your code editor. The

example may not run correctly if partia2l examples are used or if paragraph formatting is lost.

4.2.1Methods

4.2.1.1 Addnew Method Example

This example uses the AddNew method to create a new record with the specified name.

```
'BeginAddNewVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub AddNewX()

    'recordset and connection variables
  Dim Cnxn As ADODB.Connection
  Dim rstEmployees As ADODB.Recordset
  Dim strCnxn As String
  Dim strSQL As String
    'record variables
  Dim strID As String
```

```
    Dim strFirstName As String
    Dim strLastName As String
    Dim blnRecordAdded As Boolean

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Northwind;User
Id=sa;Password=;"
    Cnxn.Open strCnxn

    ' Open Employees Table with a cursor that allows updates
    Set rstEmployees = New ADODB.Recordset
    strSQL = "Employees"
    rstEmployees.Open strSQL, strCnxn, adOpenKeyset, adLockOptimistic, adCmdTable

    ' Get data from the user
    strFirstName = Trim(InputBox("Enter first name:"))
    strLastName = Trim(InputBox("Enter last name:"))

    ' Proceed only if the user actually entered something
    ' for both the first and last names
    If strFirstName <> "" And strLastName <> "" Then

        rstEmployees.AddNew
          rstEmployees!FirstName = strFirstName
          rstEmployees!LastName = strLastName
        rstEmployees.Update
        blnRecordAdded = True

        ' Show the newly added data
        MsgBox "New record: " & rstEmployees!EmployeeID & " " & _
            rstEmployees!FirstName & " " & rstEmployees!LastName

    Else
        MsgBox "Please enter a first name and last name."
    End If

    ' Delete the new record because this is a demonstration
    Cnxn.Execute "DELETE FROM Employees WHERE EmployeeID = '" & strID & "'"

     ' clean up
    rstEmployees.Close
    Cnxn.Close
    Set rstEmployees = Nothing
    Set Cnxn = Nothing

End Sub
'EndAddNewVB
```

4.2.1.2Append and CreateParameter Methods Example

This example uses the Append and CreateParameter methods to execute a stored procedure with an input parameter.

```
'BeginAppendVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string
```

```
Public Sub AppendX()

    'recordset, command and connection variables
    Dim Cnxn As ADODB.Connection
    Dim cmdByRoyalty As ADODB.Command
    Dim prmByRoyalty As ADODB.Parameter
    Dim rstByRoyalty As ADODB.Recordset
    Dim rstAuthors As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLAuthors As String
    Dim strSQLByRoyalty As String
    'record variables
    Dim intRoyalty As Integer
    Dim strAuthorID As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

    ' Open command object with one parameter
    Set cmdByRoyalty = New ADODB.Command
    cmdByRoyalty.CommandText = "byroyalty"
    cmdByRoyalty.CommandType = adCmdStoredProc

    ' Get parameter value and append parameter
    intRoyalty = Trim(InputBox("Enter royalty:"))
    Set prmByRoyalty = cmdByRoyalty.CreateParameter("percentage", adInteger,
adParamInput)
    cmdByRoyalty.Parameters.Append prmByRoyalty
    prmByRoyalty.Value = intRoyalty

    ' Create recordset by executing the command
    Set cmdByRoyalty.ActiveConnection = Cnxn
    Set rstByRoyalty = cmdByRoyalty.Execute

    ' Open the Authors Table to get author names for display
    ' and set cursor client-side
    Set rstAuthors = New ADODB.Recordset
    strSQLAuthors = "Authors"
    rstAuthors.Open strSQLAuthors, Cnxn, adUseClient, adLockOptimistic, adCmdTable

    ' Print recordset adding author names from Authors table
    Debug.Print "Authors with " & intRoyalty & " percent royalty"

    Do Until rstByRoyalty.EOF
        strAuthorID = rstByRoyalty!au_id
        Debug.Print "     " & rstByRoyalty!au_id & ", ";
        rstAuthors.Filter = "au_id = '" & strAuthorID & "'"
        Debug.Print rstAuthors!au_fname & " " & rstAuthors!au_lname
        rstByRoyalty.MoveNext
    Loop

    ' clean up
    rstByRoyalty.Close
    rstAuthors.Close
    Cnxn.Close
    Set Cnxn = Nothing
```

```
    Set rstAuthors = Nothing
    Set rstByRoyalty = Nothing


End Sub
'EndAppendVB
```

4.2.1.3AppendChunk and Getchuk Methods Example

This example uses the AppendChunk and GetChunk methods to fill an image field with data from another record.

```
'BeginAppendChunkVB

     'To integrate this code
     'replace the data source and initial catalog values
     'in the connection string

Public Sub AppendChunkX()

     'recordset and connection variables
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim rstPubInfo As ADODB.Recordset
    Dim strSQLPubInfo As String
     'record variables
    Dim strPubID As String
    Dim strPRInfo As String
    Dim lngOffset As Long
    Dim lngLogoSize As Long
    Dim varLogo As Variant
    Dim varChunk As Variant
    Dim strMsg As String

    Const conChunkSize = 100

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

    ' Open the pub_info table with a cursor that allows updates
    Set rstPubInfo = New ADODB.Recordset
    strSQLPubInfo = "pub_info"
    rstPubInfo.Open strSQLPubInfo, Cnxn, adOpenKeyset, adLockOptimistic,
adCmdTable

    ' Prompt for a logo to copy
    strMsg = "Available logos are : " & vbCr & vbCr
    Do While Not rstPubInfo.EOF
        strMsg = strMsg & rstPubInfo!pub_id & vbCr & _
            Left(rstPubInfo!pr_info, InStr(rstPubInfo!pr_info, ",") - 1) & _
            vbCr & vbCr
        rstPubInfo.MoveNext
    Loop

    strMsg = strMsg & "Enter the ID of a logo to copy:"
    strPubID = InputBox(strMsg)
```

```
    ' Copy the logo to a variable in chunks
    rstPubInfo.Filter = "pub_id = '" & strPubID & "'"
    lngLogoSize = rstPubInfo!logo.ActualSize
    Do While lngOffset < lngLogoSize
        varChunk = rstPubInfo!logo.GetChunk(conChunkSize)
        varLogo = varLogo & varChunk
        lngOffset = lngOffset + conChunkSize
    Loop

    ' Get data from the user
    strPubID = Trim(InputBox("Enter a new pub ID" & _
                                    " [must be > 9899 & < 9999]:"))
    strPRInfo = Trim(InputBox("Enter descriptive text:"))

    ' Add the new publisher to the publishers table to avoid
    ' getting an error due to foreign key constraint
    Cnxn.Execute "INSERT publishers(pub_id, pub_name) VALUES('" & _
                    strPubID & "','Your Test Publisher')"

    ' Add a new record, copying the logo in chunks
    rstPubInfo.AddNew
    rstPubInfo!pub_id = strPubID
    rstPubInfo!pr_info = strPRInfo

    lngOffset = 0 ' Reset offset
    Do While lngOffset < lngLogoSize
        varChunk = LeftB(RightB(varLogo, lngLogoSize - lngOffset), _
            conChunkSize)
        rstPubInfo!logo.AppendChunk varChunk
        lngOffset = lngOffset + conChunkSize
    Loop
    rstPubInfo.Update

     ' Show the newly added data
    MsgBox "New record: " & rstPubInfo!pub_id & vbCr & _
        "Description: " & rstPubInfo!pr_info & vbCr & _
        "Logo size: " & rstPubInfo!logo.ActualSize

    ' Delete new records because this is a demo
    rstPubInfo.Requery
    Cnxn.Execute "DELETE FROM pub_info " & _
        "WHERE pub_id = '" & strPubID & "'"

    Cnxn.Execute "DELETE FROM publishers " & _
        "WHERE pub_id = '" & strPubID & "'"

     ' clean up
    rstPubInfo.Close
    Cnxn.Close
    Set rstPubInfo = Nothing
    Set Cnxn = Nothing

End Sub
'EndAppendChunkVB
```

4.2.1.4BeginTrans, CommitTrans, and RollbackTrans Methods

Example

This example changes the book type of all psychology books in the Titles table of the

database. After the BeginTrans method starts a transaction that isolates all the changes made to the Titles table, the CommitTrans method saves the changes. You can use the RollbackTrans method to undo changes that you saved using the Update method.

```
'BeginBeginTransVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub BeginTransX()

    'recordset and connection variables
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim rstTitles As ADODB.Recordset
    Dim strSQLTitles As String
    'record variables
    Dim strTitle As String
    Dim strMessage As String

    ' Open connection
        strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Set Cnxn = New ADODB.Connection
    Cnxn.Open strCnxn

    ' Open recordset dynamic to allow for changes
    Set rstTitles = New ADODB.Recordset
    strSQLTitles = "Titles"
    rstTitles.Open strSQLTitles, Cnxn, adOpenDynamic, adLockPessimistic, adCmdTable

    Cnxn.BeginTrans

    ' Loop through recordset and prompt user
    ' to change the type for a specified title

    rstTitles.MoveFirst

    Do Until rstTitles.EOF

      If Trim(rstTitles!Type) = "psychology" Then
          strTitle = rstTitles!Title
          strMessage = "Title: " & strTitle & vbCr & _
          "Change type to self help?"

          ' If yes, change type for the specified title
          If MsgBox(strMessage, vbYesNo) = vbYes Then
              rstTitles!Type = "self_help"
              rstTitles.Update
          End If

      End If
    rstTitles.MoveNext
    Loop

    ' Prompt user to commit all changes made
    If MsgBox("Save all changes?", vbYesNo) = vbYes Then
        Cnxn.CommitTrans
    Else
```

```
        Cnxn.RollbackTrans
    End If

    ' Print recordset
    rstTitles.Requery
    rstTitles.MoveFirst
    Do While Not rstTitles.EOF
        Debug.Print rstTitles!Title & " - " & rstTitles!Type
        rstTitles.MoveNext
    Loop

    ' Restore original data as this is a demo
    rstTitles.MoveFirst

    Do Until rstTitles.EOF
        If Trim(rstTitles!Type) = "self_help" Then
            rstTitles!Type = "psychology"
            rstTitles.Update
        End If

    rstTitles.MoveNext
    Loop

     ' clean up
    rstTitles.Close
    Cnxn.Close
    Set rstTitles = Nothing
    Set Cnxn = Nothing

End Sub


'EndBeginTransVB
```

4.2.1.5Cancle Method Example

This example uses the Cancel method to cancel a command executing on a Connection object if the connection is busy.

```
'BeginCancelVB

     'To integrate this code
     'replace the data source and initial catalog values
     'in the connection string

Public Sub CancelX()

     'recordset and connection variables
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strCmdChange As String
    Dim strCmdRestore As String
     'record variables
    Dim blnChanged As Boolean

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn
```

```
    ' Define command strings
    strCmdChange = "UPDATE titles SET type = 'self_help' WHERE type = 'psychology'"
    strCmdRestore = "UPDATE titles SET type = 'psychology' " & _
                    "WHERE type = 'self_help'"

    ' Begin a transaction, then execute a command asynchronously
    Cnxn.BeginTrans
    Cnxn.Execute strCmdChange, , adAsyncExecute
    ' do something else for a little while –
    ' use i = 1 to 32000 to allow completion
      Dim i As Integer
        For i = 1 To 1000
            i = i + i
            Debug.Print i
        Next i

    ' If the command has NOT completed, cancel the execute and
    ' roll back the transaction; otherwise, commit the transaction
    If CBool(Cnxn.State And adStateExecuting) Then
        Cnxn.Cancel
        Cnxn.RollbackTrans
        blnChanged = False
        MsgBox "Update canceled."
    Else
        Cnxn.CommitTrans
        blnChanged = True
        MsgBox "Update complete."
    End If

    ' If the change was made, restore the data
    ' because this is only a demo
    If blnChanged Then
        Cnxn.Execute strCmdRestore
        MsgBox "Data restored."
    End If

      ' clean up
    Cnxn.Close
    Set Cnxn = Nothing

End Sub
'EndCancelVB
```

4.2.1.6Clone Method Example

This example uses the Clone method to create copies of a Recordset and then lets the user position the record pointer of each copy independently.

```
'BeginCloneVB

      'To integrate this code
      'replace the data source and initial catalog values
      'in the connection string

Public Sub CloneX()

      'recordset array and connection variables
    Dim arstStores(1 To 3) As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strSQLStore As String
```

```
    Dim strCnxn As String
     'record variables
    Dim intLoop As Integer
    Dim strMessage As String
    Dim strFind As String

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

    ' Open recordset as a static cursor type recordset
    Set arstStores(1) = New ADODB.Recordset
    strSQLStore = "SELECT stor_name FROM Stores ORDER BY stor_name"
    arstStores(1).Open strSQLStore, strCnxn, adOpenStatic, adLockBatchOptimistic,
adCmdText

    ' Create two clones of the original Recordset
    Set arstStores(2) = arstStores(1).Clone
    Set arstStores(3) = arstStores(1).Clone

          ' Loop through the array so that on each pass the user
          ' is searching a different copy of the same Recordset
    Do
        For intLoop = 1 To 3
             ' Ask for search string while showing where
             ' the current record pointer is for each Recordset
            strMessage = _
                "Recordsets from stores table:" & vbCr & _
                "    1 - Original - Record pointer at " & arstStores(1)!stor_name & vbCr & _
                "    2 - Clone - Record pointer at " & arstStores(2)!stor_name & vbCr & _
                "    3 - Clone - Record pointer at " & arstStores(3)!stor_name & vbCr & _
                "Enter search string for #" & intLoop & ":"

            strFind = Trim(InputBox(strMessage))
             ' make sure something was entered, if not then EXIT loop
            If strFind = "" Then Exit Do

             ' otherwise locate the record from the entered string
            arstStores(intLoop).Filter = "stor_name = '" & strFind & "'"

             'if there's no match, jump to the last record
            If arstStores(intLoop).EOF Then
                arstStores(intLoop).Filter = adFilterNone
                arstStores(intLoop).MoveLast
            Else
                MsgBox "Found " & strFind
            End If
        Next intLoop
    Loop

     ' clean up
    arstStores(1).Close
    arstStores(2).Close
    arstStores(3).Close
    Cnxn.Close
    Set arstStores(1) = Nothing
    Set arstStores(2) = Nothing
```

```
    Set arstStores(3) = Nothing
    Set Cnxn = Nothing

End Sub
'EndCloneVB
```

4.2.1.7Compare Bookmarks Method Example

This example demonstrates the CompareBookmarks method. The relative value of bookmarks is seldom needed unless a particular bookmark is somehow special.
Designate a random row of a Recordset derived from the Authors table as the target of a search. Then display the position of each row relative to that target.

```
'BeginCompareBookmarksVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub CompareBookmarksX()

     ' recordset and connection variables
    Dim rstAuthors As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strSQLAuthors As String
    Dim strCnxn As String

     ' comparison variables
    Dim Count As Integer
    Dim target As Variant
    Dim result As Long
    Dim strAnswer As String
    Dim strTitle As String
    strTitle = "CompareBookmarks Example"

  ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

     ' Open recordset as a static cursor type recordset
    Set rstAuthors = New ADODB.Recordset
    strSQLAuthors = "SELECT * FROM Authors"
    rstAuthors.Open strSQLAuthors, Cnxn, adOpenStatic, adLockReadOnly, adCmdText

    Count = rstAuthors.RecordCount
    Debug.Print "Rows in the Recordset = "; Count

     ' Exit if an empty recordset
    If Count = 0 Then Exit Sub

     ' Get position between 0 and count -1
    Randomize
    Count = (Int(Count * Rnd))
    Debug.Print "Randomly chosen row position = "; Count
     ' Move row to random position
    rstAuthors.Move Count, adBookmarkFirst
     ' Remember the mystery row
    target = rstAuthors.Bookmark
```

```
    Count = 0
    rstAuthors.MoveFirst
          ' Loop through recordset
    Do Until rstAuthors.EOF
        result = rstAuthors.CompareBookmarks(rstAuthors.Bookmark, target)

        If result = adCompareNotEqual Then
            Debug.Print "Row "; Count; ": Bookmarks are not equal."
        ElseIf result = adCompareNotComparable Then
            Debug.Print "Row "; Count; ": Bookmarks are not comparable."
        Else
            Select Case result
                Case adCompareLessThan
                    strAnswer = "less than"
                Case adCompareEqual
                    strAnswer = "equal to"
                Case adCompareGreaterThan
                    strAnswer = "greater than"
                Case Else
                    strAnswer = "in error comparing to"
            End Select
               'show the results row-by-row
            Debug.Print "Row position " & Count & " is " & strAnswer & " the target."
        End If

        Count = Count + 1
        rstAuthors.MoveNext
    Loop

     ' clean up
    rstAuthors.Close
    Cnxn.Close
    Set rstAuthors = Nothing
    Set Cnxn = Nothing

End Sub
'EndCompareBookmarksVB
```

4.2.1.8ConvertToString Method Example

```
'BeginConvertToStringVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub ConvertToStringX()

     ' to integrate this code replace the server name
     ' in the CreateObject call

     ' RDS variables
    Dim rdsDS As RDS.DataSpace
    Dim rdsDC As RDS.DataControl
    Dim rdsDF As Object
     ' recordset and connection variables
    Dim rsAuthors As ADODB.Recordset
    Dim strSQLAuthors As String
```

```
    Dim strCnxn As String
    Dim varString As Variant

     ' Create a DataSpace object
    Set rdsDS = New RDS.DataSpace
     ' Create a DataFactory object
    Set rdsDF = rdsDS.CreateObject("RDSServer.DataFactory", "http://MyServer")

     ' Get all of the Author records
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=pubs;User
ID=sa;Password=;"
    strSQLAuthors = "SELECT * FROM Authors"
    Set rsAuthors = rdsDF.Query(strCnxn, strSQLAuthors)
     ' Show results
    Debug.Print "Old RDS recordset:" & rsAuthors.RecordCount

     ' Convert the recordset into a MIME formatted string
    varString = rdsDF.ConvertToString(rsAuthors)

     ' Convert string value back into an ADO Recordset
    Set rdsDC = New RDS.DataControl
    rdsDC.SQL = varString
    rdsDC.ExecuteOptions = adcExecSync
    rdsDC.FetchOptions = adcFetchUpFront
    rdsDC.Refresh
     ' Show results
    Debug.Print "New ADO recordset:" & rdsDC.Recordset.RecordCount

     ' clean up
    rsAuthors.Close
    Set rsAuthors = Nothing
    Set rdsDC = Nothing
    Set rdsDS = Nothing
    Set rdsDC = Nothing

End Sub
'EndConvertToStringVB
```

4.2.1.9CopyRecord, CopyTo, and SaveToFile Methods Example

This example demonstrates how to create copies of a file using Stream or Record objects. One copy is made to a Web folder for Internet publishing. Other properties and methods shown include Stream Type, Open, LoadFromFile, and Record Open.

```
'BeginCopyRecordVB
Option Explicit

Public Sub CopyRecordX()
    ' Declare variables
    Dim strPicturePath, strStreamPath, strStream2Path, _
        strRecordPath, strStreamURL, strRecordURL As String
    Dim objStream, objStream2 As Stream
    Dim objRecord As Record
    Dim objField As Field

    ' Instantiate objects
    Set objStream = New Stream
    Set objStream2 = New Stream
    Set objRecord = New Record
```

```
    ' Initialize path and URL strings
    strPicturePath = _
    "C:\Program Files\Microsoft Office\Clipart\Popular\Checkmrk.wmf"
    strStreamPath = "\\websrv\folder\mywmf.wmf"
    strStreamURL = "URL=http://websrv/folder/mywmf.wmf"
    strStream2Path = "D:\samples\check2.wmf"
    strRecordPath = "\\websrv\folder\mywmf.wmf"
    strRecordURL = "http://websrv/folder/mywmf.wmf"

    ' Load the file into the stream
    objStream.Open
    objStream.Type = adTypeBinary
    objStream.LoadFromFile (strPicturePath)

    ' Save the stream to a new path and filename
    objStream.SaveToFile strStreamPath, adSaveCreateOverWrite

    ' Copy the contents of the first stream to a second stream
    objStream2.Open
    objStream2.Type = adTypeBinary
    objStream.CopyTo objStream2

    ' Save the second stream to a different path
    objStream2.SaveToFile strStream2Path, adSaveCreateOverWrite

    ' Because strStreamPath is a Web Folder, open a Record on the URL
    objRecord.Open "", strStreamURL

    ' Display the Fields of the record
    For Each objField In objRecord.fields
        Debug.Print objField.Name & ": " & objField.Value
    Next

    ' Copy the record to a new URL
    objRecord.CopyRecord "", strRecordURL, , , adCopyOverWrite

    ' Load each copy of the graphic into Image controls for viewing
    Image1.Picture = LoadPicture(strPicturePath)
    Image2.Picture = LoadPicture(strStreamPath)
    Image3.Picture = LoadPicture(strStream2Path)
    Image4.Picture = LoadPicture(strRecordPath)

    ' Clean up
    objStream.Close
    objStream2.Close
    objRecord.Close
End Sub
'EndCopyRecordVB
```

4.2.1.10CreateRecordset Method Example

You can create a Recordset object and specify the column information. You can then insert data into the Recordset object; the underlying rowset buffers the inserts.

The following code example shows how to define a Recordset by using the RDSServer.DataFactory object. You can also do this with the RDS.DataControl object.

```
'BeginRsDefineShapeVB
Sub RsDefineShape()

    Dim ADF As RDSServer.DataFactory
```

```
Dim vntRecordShape(3)
Dim vntField1Shape(3)
Dim vntField2Shape(3)
Dim vntField3Shape(3)
Dim vntField4Shape(3)

Set ADF = New RDSServer.DataFactory

' For each field, specify the name,
' type, size, and nullability.

vntField1Shape(0) = "Name"      ' Column name.
vntField1Shape(1) = CInt(129)     ' Column type.
vntField1Shape(2) = CInt(40)     ' Column size.
vntField1Shape(3) = False          ' Nullable?

vntField2Shape(0) = "Age"
vntField2Shape(1) = CInt(3)
vntField2Shape(2) = CInt(-1)
vntField2Shape(3) = True

vntField3Shape(0) = "DateOfBirth"
vntField3Shape(1) = CInt(7)
vntField3Shape(2) = CInt(-1)
vntField3Shape(3) = True

vntField4Shape(0) = "Balance"
vntField4Shape(1) = CInt(6)
vntField4Shape(2) = CInt(-1)
vntField4Shape(3) = True

' Put all fields into an array of arrays.
vntRecordShape(0) = vntField1Shape
vntRecordShape(1) = vntField2Shape
vntRecordShape(2) = vntField3Shape
vntRecordShape(3) = vntField4Shape

' Use the RDSServer.DataFactory to create an empty
' recordset. It takes an array of variants where
' every element is itself another array of
' variants, one for every column required in the
' recordset.
' The elements of the inner array are the column's
' name, type, size, and nullability.
'
' NOTE: You could just use the RDS.DataControl object
' instead of the RDSServer.DataFactory object. In
' that case, the following code would be Set NewRS
' = ADC1.CreateRecordset(vntRecordShape)
Dim NewRs As ADODB.Recordset
Set NewRs = ADF.CreateRecordSet(vntRecordShape)

Dim fields(3)
fields(0) = vntField1Shape(0)
fields(1) = vntField2Shape(0)
fields(2) = vntField3Shape(0)
fields(3) = vntField4Shape(0)

' Populate the new recordset with data values.
```

```
    Dim fieldVals(3)

    ' Use AddNew to add the records.
    fieldVals(0) = "Joe"
    fieldVals(1) = 5
    fieldVals(2) = CDate(#1/5/1996#)
    fieldVals(3) = 123.456
    NewRs.AddNew fields, fieldVals

    fieldVals(0) = "Mary"
    fieldVals(1) = 6
    fieldVals(2) = CDate(#6/5/1996#)
    fieldVals(3) = 31
    NewRs.AddNew fields, fieldVals

    fieldVals(0) = "Alex"
    fieldVals(1) = 13
    fieldVals(2) = CDate(#1/6/1996#)
    fieldVals(3) = 34.0001
    NewRs.AddNew fields, fieldVals

    fieldVals(0) = "Susan"
    fieldVals(1) = 13
    fieldVals(2) = CDate(#8/6/1996#)
    fieldVals(3) = 0#
    NewRs.AddNew fields, fieldVals

    NewRs.MoveFirst

    ' Set the newly created and populated Recordset to
    ' the SourceRecordset property of the
    ' RDS.DataControl to bind to visual controls
    Dim ADC1 As RDS.DataControl
    Set ADC1 = New RDS.DataControl
    Set ADC1.SourceRecordset = NewRs

End Sub
'EndRsDefineShapeVB
```

4.2.1.11Delete Method Example

This example uses the Delete method to remove a specified record from a Recordset

```
'BeginDeleteVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub DeleteX()

    Dim rstRoySched As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLRoySched As String

    Dim strMsg As String
    Dim strTitleID As String
    Dim intLoRange As Integer
    Dim intHiRange As Integer
```

```
    ' reset the stream object
    objStream.Flush
    objStream.Close
    rcFile.Close

    ' reopen record to see new contents of text file
    rcFile.Open strFile, Cnxn, adModeReadWrite, adOpenIfExists Or
adCreateNonCollection
    objStream.Open rcFile, adModeReadWrite, adOpenStreamFromRecord

    Debug.Print "Source: " & rcFile.Source
    Debug.Print "Edited text: " & objStream.ReadText

    ' copy the file to another folder
    strDestFile = "test2/test.txt"
    rcFile.CopyRecord strFile, strDestFile, , , adCopyOverWrite

    ' delete the original file
    rcFile.DeleteRecord

    ' move the file from the subfolder back to original location
    rcDestFolder.Open strDestFolder, Cnxn, adOpenIfExists Or adCreateCollection
    Set rsDestFolder = rcDestFolder.GetChildren
    rsDestFolder.MoveFirst

    ' position current record at on the correct file
    Do While Not rsDestFolder.EOF Or rsDestFolder(0) = "test1.txt"
        rsDestFolder.MoveNext
    Loop

    ' open a record on the correct row of the recordset
    rcDestFile.Open rsDestFolder, Cnxn

    ' do the move
    rcDestFile.MoveRecord strDestFile, strFile, , , adMoveOverWrite

End Sub
'EndDeleteRecordVB
```

4.2.1.13Execute, Requery, and Clear Methods Example

This example demonstrates the Execute method when run from both a Command object and a Connection object. It also uses the Requery method to retrieve current data in a Recordset, and the Clear method to clear the contents of the Errors collection. (The Errors collection is accessed via the Connection object of the ActiveConnection property of the Recordset.) The ExecuteCommand and PrintOutput procedures are required for this procedure to run.

```
'BeginExecuteVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub ExecuteX()

    ' connection, command, and recordset variables
    Dim Cnxn As ADODB.Connection
    Dim cmdChange As ADODB.Command
    Dim rstTitles As ADODB.Recordset
```

```
End Sub
'EndDeleteVB
```

4.2.1.12DeleteRecord and MoveRecord Methods Example

This example demonstrates how to copy, move, edit, and delete the contents of a text file published to a Web folder. Other properties and methods used include GetChildren, ParentURL, Source, and Flush.

```
'BeginDeleteRecordVB

    'to use this code replace all connection
    'and file/folder variables

Public Sub DeleteRecordX()

     ' connection and recordset variables
    Dim rsDestFolder As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String

     ' file as record variables
    Dim rcFile As ADODB.Record
    Dim rcDestFile As ADODB.Record
    Dim rcDestFolder As ADODB.Record
    Dim objStream As Stream

     ' file variables
    Dim strFile As String
    Dim strDestFile As String
    Dim strDestFolder As String

     ' instantiate variables
    Set rsDestFolder = New ADODB.Recordset
    Set rcDestFolder = New ADODB.Record
    Set rcFile = New ADODB.Record
    Set rcDestFile = New ADODB.Record
    Set objStream = New ADODB.Stream

     ' open a record on a text file
    Set Cnxn = New ADODB.Connection
    strCnxn = "url=http://a-dgayne2/"
    Cnxn.Open strCnxn
    strFile = "test/test2.txt"
    rcFile.Open strFile, Cnxn, adModeReadWrite, adOpenIfExists Or
adCreateNonCollection

     ' edit the contents of the text file
    objStream.Open rcFile, , adOpenStreamFromRecord

    Debug.Print "Source: " & strCnxn & rcFile.Source
    Debug.Print "Original text: " & objStream.ReadText

    objStream.Position = 0
    objStream.WriteText "Newer Text"
    objStream.Position = 0

    Debug.Print "New text: " & objStream.ReadText
```

```
    ' reset the stream object
    objStream.Flush
    objStream.Close
    rcFile.Close

    ' reopen record to see new contents of text file
    rcFile.Open strFile, Cnxn, adModeReadWrite, adOpenIfExists Or
adCreateNonCollection
    objStream.Open rcFile, adModeReadWrite, adOpenStreamFromRecord

    Debug.Print "Source: " & rcFile.Source
    Debug.Print "Edited text: " & objStream.ReadText

    ' copy the file to another folder
    strDestFile = "test2/test.txt"
    rcFile.CopyRecord strFile, strDestFile, , , adCopyOverWrite

    ' delete the original file
    rcFile.DeleteRecord

    ' move the file from the subfolder back to original location
    rcDestFolder.Open strDestFolder, Cnxn, adOpenIfExists Or adCreateCollection
    Set rsDestFolder = rcDestFolder.GetChildren
    rsDestFolder.MoveFirst

    ' position current record at on the correct file
    Do While Not rsDestFolder.EOF Or rsDestFolder(0) = "test1.txt"
        rsDestFolder.MoveNext
    Loop

    ' open a record on the correct row of the recordset
    rcDestFile.Open rsDestFolder, Cnxn

    ' do the move
    rcDestFile.MoveRecord strDestFile, strFile, , , adMoveOverWrite

End Sub
'EndDeleteRecordVB
```

4.2.1.13Execute, Requery, and Clear Methods Example

This example demonstrates the Execute method when run from both a Command object and a Connection object. It also uses the Requery method to retrieve current data in a Recordset, and the Clear method to clear the contents of the Errors collection. (The Errors collection is accessed via the Connection object of the ActiveConnection property of the Recordset.) The ExecuteCommand and PrintOutput procedures are required for this procedure to run.

```
'BeginExecuteVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub ExecuteX()

    ' connection, command, and recordset variables
    Dim Cnxn As ADODB.Connection
    Dim cmdChange As ADODB.Command
    Dim rstTitles As ADODB.Recordset
```

```
    Dim Err As ADODB.Error
    Dim strSQLChange As String
    Dim strSQLRestore As String
    Dim strSQLTitles
    Dim strCnxn As String

     ' Define two SQL statements to execute as command text
    strSQLChange = "UPDATE Titles SET Type = 'self_help' WHERE Type = 'psychology'"
    strSQLRestore = "UPDATE Titles SET Type = 'psychology' WHERE Type = 'self_help'"

     ' Open connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Set Cnxn = New ADODB.Connection
    Cnxn.Open strCnxn

     ' Create command object
    Set cmdChange = New ADODB.Command
    Set cmdChange.ActiveConnection = Cnxn
    cmdChange.CommandText = strSQLChange

     ' Open titles table
    Set rstTitles = New ADODB.Recordset
    strSQLTitles = "titles"
    rstTitles.Open strSQLTitles, Cnxn, , , adCmdTable

    ' Print report of original data
    Debug.Print _
        "Data in Titles table before executing the query"
    PrintOutput rstTitles

    ' Clear extraneous errors from the Errors collection
    Cnxn.Errors.Clear

    ' Call the ExecuteCommand subroutine below to execute cmdChange command
    ExecuteCommand cmdChange, rstTitles

    ' Print report of new data
    Debug.Print _
        "Data in Titles table after executing the query"
    PrintOutput rstTitles

    ' Use the Connection object's execute method to
    ' execute SQL statement to restore data and trap for
    ' errors, checking the Errors collection if necessary
    On Error GoTo Err_Execute
    Cnxn.Execute strSQLRestore, , adExecuteNoRecords
    On Error GoTo 0

    ' Retrieve the current data by requerying the recordset
    rstTitles.Requery

    ' Print report of restored data using sub from below
    Debug.Print "Data after executing the query to restore the original information "
    PrintOutput rstTitles

    rstTitles.Close
    Cnxn.Close
    Set rstTitles = Nothing
```

```
    Set Cnxn = Nothing

    Exit Sub

Err_Execute:

    ' Notify user of any errors that result from
    ' executing the query
    If rstTitles.ActiveConnection.Errors.Count >= 0 Then
        For Each Err In rstTitles.ActiveConnection.Errors
            MsgBox "Error number: " & Err.Number & vbCr & _
                Err.Description
        Next Err
    End If
    Resume Next

End Sub

Public Sub ExecuteCommand(cmdTemp As ADODB.Command, rstTemp As
ADODB.Recordset)

    Dim Err As Error

    ' Run the specified Command object and trap for
    ' errors, checking the Errors collection
    On Error GoTo Err_Execute
    cmdTemp.Execute
    On Error GoTo 0

    ' Retrieve the current data by requerying the recordset
    rstTemp.Requery

    Exit Sub

Err_Execute:

    ' Notify user of any errors that result from
    ' executing the query
    If rstTemp.ActiveConnection.Errors.Count > 0 Then
        For Each Err In rstTemp.ActiveConnection.Errors
            MsgBox "Error number: " & Err.Number & vbCr & _
                Err.Description
        Next Err
    End If

    Resume Next

End Sub

Public Sub PrintOutput(rstTemp As ADODB.Recordset)

    ' Enumerate Recordset
    Do While Not rstTemp.EOF
        Debug.Print "    " & rstTemp!Title & _
            ", " & rstTemp!Type
        rstTemp.MoveNext
    Loop

End Sub
```

```
'EndExecuteVB
```

4.2.1.14Find Method Example

This example uses the Recordset object's Find method to locate and count the number of business titles in the Pubs database. The example assumes the underlying provider does not support similar functionality

```
'BeginFindVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub FindX()

     ' connection and recordset variables
    Dim Cnxn As New ADODB.Connection
    Dim rstTitles As New ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLTitles As String

     ' record variables
    Dim mark As Variant
    Dim count As Integer

     ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
    Cnxn.Open strCnxn

    ' open recordset with default parameters which are
    ' sufficient to search forward through a Recordset
    Set rstTitles = New ADODB.Recordset
    strSQLTitles = "SELECT title_id FROM titles"
    rstTitles.Open strSQLTitles, Cnxn, adOpenStatic, adLockReadOnly, adCmdText

    count = 0
    rstTitles.Find "title_id LIKE 'BU%'"

    Do While Not rstTitles.EOF
        'continue if last find succeeded
        Debug.Print "Title ID: "; rstTitles!title_id
        'count the last title found
        count = count + 1
        ' note current position
        mark = rstTitles.Bookmark
        rstTitles.Find "title_id LIKE 'BU%'", 1, adSearchForward, mark
        ' above code skips current record to avoid finding the same row repeatedly;
        ' last arg (bookmark) is redundant because Find searches from current position
    Loop

    Debug.Print "The number of business titles is " & count

     ' clean up
    rstTitles.Close
    Cnxn.Close
    Set rstTitles = Nothing
    Set Cnxn = Nothing
```

```
End Sub
'EndFindVB
```

4.2.1.15GetRows Method Example

This example uses the GetRows method to retrieve a specified number of rows from a Recordset and to fill an array with the resulting data. The GetRows method will return fewer than the desired number of rows in two cases: either if EOF has been reached, or if GetRows tried to retrieve a record that was deleted by another user. The function returns False only if the second case occurs. The GetRowsOK function is required for this procedure to run.

```
'BeginGetRowsVB
```

4.2.1.16GetString Method Example

This example demonstrates the GetString method.

Assume you are debugging a data access problem and want a quick, simple way of printing the current contents of a small Recordset.

```
'BeginGetStringVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub GetStringX()

     ' connection variables
    Dim Cnxn As ADODB.Connection
    Dim rstAuthors As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLAuthors As String
    Dim varOutput As Variant

     ' specific variables
    Dim strPrompt As String
    Dim strState As String

     ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

     ' get user input
    strPrompt = "Enter a state (CA, IN, KS, MD, MI, OR, TN, UT): "
    strState = Trim(InputBox(strPrompt, "GetString Example"))

     ' open recordset
    Set rstAuthors = New ADODB.Recordset
    strSQLAuthors = "SELECT au_fname, au_lname, address, city FROM Authors " & _
                    "WHERE state = '" & strState & "'"
    rstAuthors.Open strSQLAuthors, Cnxn, adOpenStatic, adLockReadOnly, adCmdText

    If Not rstAuthors.EOF Then
    ' Use all defaults: get all rows, TAB as column delimiter,
    ' CARRIAGE RETURN as row delimiter, EMPTY-string as null delimiter
        varOutput = rstAuthors.GetString(adClipString)
         ' print output
```

```
        Debug.Print "State = '" & strState & "'"
        Debug.Print "Name                Address                City" & vbCr
        Debug.Print varOutput
    Else
        Debug.Print "No rows found for state = '" & strState & "'" & vbCr
    End If

     ' clean up
    rstAuthors.Close
    Cnxn.Close
    Set rstAuthors = Nothing
    Set Cnxn = Nothing

End Sub
'EndGetStringVB
```

4.2.1.17SkipLine Method, EOS, and LineSeparator Properties Example

This example demonstrates how to manipulate text streams one line at a time. The effect of changing the line separator from the default carriage return/linefeed (adCRLF) to simply linefeed (adLF) or carriage return (adCR) is shown.

```
'BeginSkipLineVB
Public Sub SkipLineX()
    'Declare variables
    Dim i As Integer
    Dim objStream As Stream
    Dim strLine, strChar As String

    'Instantiate and open stream
    Set objStream = New Stream
    objStream.Open

    'Set line separator to line feed
    objStream.LineSeparator = adLF

    'Load text content of list box into stream
    'One line at a time
    For i = 0 To (List1.ListCount - 1)
        objStream.WriteText List1.List(i), adWriteLine
    Next

    'Display the entire stream
    Debug.Print "Whole Stream:"
    objStream.Position = 0
    Debug.Print objStream.ReadText

    'Display the first line
    Debug.Print "First Line:"
    objStream.Position = 0
    strLine = objStream.ReadText(adReadLine)
    Debug.Print strLine
    Debug.Print "Line length: " + Str(Len(strLine))

    'Skip a line, then display another line
    Debug.Print "Third Line:"
    objStream.SkipLine
    strLine = objStream.ReadText(adReadLine)
    Debug.Print strLine
```

```
        Debug.Print "Line length: " + Str(Len(strLine))

        'Switch line separator to carriage return
        'All items from list will be considered one line
        'Assuming no CRs have been loaded into stream
        Debug.Print "Whole Stream/First Line:"
        objStream.Position = 0
        objStream.LineSeparator = adCR
        strLine = objStream.ReadText(adReadLine)
        Debug.Print strLine
        Debug.Print "Line length: " + Str(Len(strLine))
        Debug.Print "Stream size: " + Str(objStream.Size)

        'Use EOS to Determine End of Stream
        Debug.Print "Character by character:"
        objStream.Position = 0
        Do Until objStream.EOS
            strChar = objStream.ReadText(1)
            Debug.Print strChar
        Loop
End Sub
'EndSkipLineVB
```

4.2.1.18Move Method Example

This example uses the Move method to position the record pointer based on user input.

```
'BeginMoveVB

        'To integrate this code
        'replace the data source and initial catalog values
        'in the connection string

Public Sub MoveX()

        ' connection and recordset variables
    Dim rstAuthors As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLAuthors As String
        ' record variables
    Dim varBookmark As Variant
    Dim strCommand As String
    Dim lngMove As Long

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=a-iresmi2000;Initial Catalog=pubs;User Id=sa;Password=;"
    Cnxn.Open strCnxn

    ' Open recordset from Authors table
    Set rstAuthors = New ADODB.Recordset
    rstAuthors.CursorLocation = adUseClient
    ' Use client cursor to allow use of AbsolutePosition property
    strSQLAuthors = "SELECT au_id, au_fname, au_lname, city, state FROM Authors ORDER BY au_lname"
    rstAuthors.Open strSQLAuthors, strCnxn, adOpenStatic, adLockOptimistic, adCmdText
```

```
    rstAuthors.MoveFirst

    Do
        ' Display information about current record and
        ' ask how many records to move

        strCommand = InputBox( _
            "Record " & rstAuthors.AbsolutePosition & _
            " of " & rstAuthors.RecordCount & vbCr & _
            "Author: " & rstAuthors!au_fname & _
            " " & rstAuthors!au_lname & vbCr & _
            "Location: " & rstAuthors!city & _
            ", " & rstAuthors!State & vbCr & vbCr & _
            "Enter number of records to Move " & _
            "(positive or negative).")

           ' this is for exiting the loop
        If strCommand = "" Then Exit Do

        ' Store bookmark in case the Move goes too far
        ' forward or backward
        varBookmark = rstAuthors.Bookmark

        ' Move method requires parameter of data type Long
        lngMove = CLng(strCommand)
        rstAuthors.Move lngMove

        ' Trap for BOF or EOF
        If rstAuthors.BOF Then
            MsgBox "Too far backward! Returning to current record."
            rstAuthors.Bookmark = varBookmark
        End If
        If rstAuthors.EOF Then
            MsgBox "Too far forward!    Returning to current record."
            rstAuthors.Bookmark = varBookmark
        End If
    Loop

     ' clean up
    rstAuthors.Close
    Cnxn.Close
    Set rstAuthors = Nothing
    Set Cnxn = Nothing

End Sub
'EndMoveVB
```

4.2.1.19MoveFirst, MoveLast, MoveNext, and MovePrevious Methods Example

This example uses the MoveFirst, MoveLast, MoveNext, and MovePrevious methods to move the record pointer of a Recordset based on the supplied command. The MoveAny procedure is required for this procedure to run.

```
'BeginMoveFirstVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string
```

```
Public Sub MoveFirstX()

    ' connection and recordset variables
    Dim rstAuthors As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLAuthors
    ' record variables
    Dim strMessage As String
    Dim intCommand As Integer

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=pubs;User
Id=sa;Password=;"
    Cnxn.Open strCnxn

    ' Open recordset from Authors table
    Set rstAuthors = New ADODB.Recordset
    rstAuthors.CursorLocation = adUseClient
    ' Use client cursor to enable AbsolutePosition property
    strSQLAuthors = "Authors"
    rstAuthors.Open strSQLAuthors, Cnxn, adOpenStatic, adLockReadOnly, adCmdTable

    ' Show current record information and get user's method choice
    Do

        strMessage = "Name: " & rstAuthors!au_fname & " " & _
            rstAuthors!au_lname & vbCr & "Record " & _
            rstAuthors.AbsolutePosition & " of " & _
            rstAuthors.RecordCount & vbCr & vbCr & _
            "[1 - MoveFirst, 2 - MoveLast, " & vbCr & _
            "3 - MoveNext, 4 - MovePrevious]"
        intCommand = Val(Left(InputBox(strMessage), 1))

        ' for exiting the loop
        If intCommand < 1 Or intCommand > 4 Then Exit Do

     ' Use specified method while trapping for BOF and EOF
     Select Case intCommand
         Case 1
             rstAuthors.MoveFirst
         Case 2
             rstAuthors.MoveLast
         Case 3
             rstAuthors.MoveNext
             If rstAuthors.EOF Then
                 MsgBox "Already at end of recordset!"
                 rstAuthors.MoveLast
             End If
         Case 4
             rstAuthors.MovePrevious
             If rstAuthors.BOF Then
                 MsgBox "Already at beginning of recordset!"
                 rstAuthors.MoveFirst
             End If
      End Select

    Loop
```

```
    ' clean up
    rstAuthors.Close
    Cnxn.Close
    Set Cnxn = Nothing
    Set rstAuthors = Nothing

End Sub

'EndMoveFirstVB
```

4.2.1.20NextRecordset Method Example

This example uses the NextRecordset method to view the data in a recordset that uses a compound command statement made up of three separate SELECT statements.

```
'BeginNextRecordsetVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub NextRecordsetX()

    ' connection and recordset variables
    Dim rstCompound As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim SQLCompound As String

    Dim intCount As Integer

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=pubs;User
Id=sa;Password=;"
    Cnxn.Open strCnxn

    ' Open compound recordset
    Set rstCompound = New ADODB.Recordset
    SQLCompound = "SELECT * FROM Authors; " & _
        "SELECT * FROM stores; " & _
        "SELECT * FROM jobs"
    rstCompound.Open SQLCompound, Cnxn, adOpenStatic, adLockReadOnly,
adCmdText

    ' Display results from each SELECT statement
    intCount = 1
    Do Until rstCompound Is Nothing
        Debug.Print "Contents of recordset #" & intCount

        Do Until rstCompound.EOF
            Debug.Print , rstCompound.Fields(0), rstCompound.Fields(1)
            rstCompound.MoveNext
        Loop

        Set rstCompound = rstCompound.NextRecordset
        intCount = intCount + 1
    Loop
```

```
    ' clean up
    rstCompound.Close
    Cnxn.Close
    Set rstCompound = Nothing
    Set Cnxn = Nothing

End Sub
'EndNextRecordsetVB
```

4.2.1.21Open and Close Methods Example

This example uses the Open and Close methods on both Recordset and Connection objects that have been opened.

```
'BeginOpenVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub OpenX()

    Dim Cnxn As ADODB.Connection
    Dim rstEmployees As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLEmployees As String
    Dim varDate As Variant

    ' Open connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
    Set Cnxn = New ADODB.Connection
    Cnxn.Open strCnxn

    ' Open employee table
    Set rstEmployees = New ADODB.Recordset
    strSQLEmployees = "employee"
    rstEmployees.Open strSQLEmployees, Cnxn, adOpenKeyset, adLockOptimistic, adCmdTable

    ' Assign the first employee record's hire date
    ' to a variable, then change the hire date
    varDate = rstEmployees!hire_date
    Debug.Print "Original data"
    Debug.Print "    Name - Hire Date"
    Debug.Print "    " & rstEmployees!fname & " " & _
        rstEmployees!lname & " - " & rstEmployees!hire_date
    rstEmployees!hire_date = #1/1/1900#
    rstEmployees.Update
    Debug.Print "Changed data"
    Debug.Print "    Name - Hire Date"
    Debug.Print "    " & rstEmployees!fname & " " & _
        rstEmployees!lname & " - " & rstEmployees!hire_date

    ' Requery Recordset and reset the hire date
    rstEmployees.Requery
    rstEmployees!hire_date = varDate
    rstEmployees.Update
    Debug.Print "Data after reset"
    Debug.Print "    Name - Hire Date"
```

```
    Debug.Print "    " & rstEmployees!fname & " " & _
        rstEmployees!lname & " - " & rstEmployees!hire_date

    rstEmployees.Close
    Cnxn.Close
    Set rstEmployees = Nothing
    Set Cnxn = Nothing

End Sub
'EndOpenVB
```

4.2.1.22OpenSchema Method Example

This example uses the OpenSchema method to display the name and type of each table in the Pubs database.

```
'BeginOpenSchemaVB

     'To integrate this code
     'replace the data source and initial catalog values
     'in the connection string

Public Sub OpenSchemaX()

    Dim Cnxn As ADODB.Connection
    Dim rstSchema As ADODB.Recordset
    Dim strCnxn As String

    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

    Set rstSchema = Cnxn.OpenSchema(adSchemaTables)

    Do Until rstSchema.EOF
        Debug.Print "Table name: " & _
            rstSchema!TABLE_NAME & vbCr & _
            "Table type: " & rstSchema!TABLE_TYPE & vbCr
        rstSchema.MoveNext
    Loop

    ' clean up
    rstSchema.Close
    Cnxn.Close
    Set rstSchema = Nothing
    Set Cnxn = Nothing

End Sub
'EndOpenSchemaVB
```

This example specifies a TABLE_TYPE query constraint in the OpenSchema method Criteria argument. As a result, only schema information for the Views specified in the Pubs database are returned. The example then displays the name(s) and type(s) of each table(s).

```
'BeginOpenSchema2VB
Public Sub OpenSchemaX2()

    Dim Cnxn2 As ADODB.Connection
    Dim rstSchema As ADODB.Recordset
```

```
    Dim strCnxn As String

    Set Cnxn2 = New ADODB.Connection
        strCnxn = "Provider=sqloledb;" & _
        "Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
    Cnxn2.Open strCnxn

    Set rstSchema = Cnxn2.OpenSchema(adSchemaTables, Array(Empty, Empty, Empty,
"VIEW"))

        Do Until rstSchema.EOF
            Debug.Print "Table name: " & _
                rstSchema!TABLE_NAME & vbCr & _
                "Table type: " & rstSchema!TABLE_TYPE & vbCr
            rstSchema.MoveNext
        Loop
    rstSchema.Close

    Cnxn2.Close

End Sub
'EndOpenSchema2VB
```

4.2.1.23Read, ReadText, Write, and WriteText Methods Example

This example demonstrates how to read the contents of a text box into both a text Stream and a binary Stream. Other properties and methods shown include Position, Size, Charset, and SetEOS.

```
'BeginReadVB
Public Sub ReadX()
    'Declare variables
    Dim objStream As Stream
    Dim varA As Variant
    Dim bytA() As Byte
    Dim i As Integer
    Dim strBytes As String

    'Instantiate and Open Stream
    Set objStream = New Stream
    objStream.Open

    'Write the text content of a textbox to the stream
    objStream.WriteText Text1.Text

    'Display the text contents and size of the stream
    objStream.Position = 0
    Debug.Print "Default text:"
    Debug.Print objStream.ReadText
    Debug.Print objStream.Size

    'Switch character set and display
    objStream.Position = 0
    objStream.Charset = "Windows-1252"
    Debug.Print "New Charset text:"
    Debug.Print objStream.ReadText
    Debug.Print objStream.Size

    'Switch to a binary stream and display
    objStream.Position = 0
```

```
    objStream.Type = adTypeBinary
    Debug.Print "Binary:"
    Debug.Print objStream.Read
    Debug.Print objStream.Size

    'Load an array of bytes with the text box text
    ReDim bytA(Len(Text1.Text))
    For i = 1 To Len(Text1.Text)
        bytA(i - 1) = CByte(Asc(Mid(Text1.Text, i, 1)))
    Next

    'Write the buffer to the binary stream and display
    objStream.Position = 0
    objStream.Write bytA()
    objStream.SetEOS
    objStream.Position = 0
    Debug.Print "Binary after Write:"
    Debug.Print objStream.Read
    Debug.Print objStream.Size

    'Switch back to a text stream and display
    Debug.Print "Translated back:"
    objStream.Position = 0
    objStream.Type = adTypeText
    Debug.Print objStream.ReadText
    Debug.Print objStream.Size
End Sub
'EndReadVB
```

4.2.1.24Refresh Method Example

This example demonstrates using the Refresh method to refresh the Parameters collection for a stored procedure Command object.

```
'BeginRefreshVB
Public Sub RefreshX()

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection strings

    ' connection and recordset variables
    Dim Cnxn As ADODB.Connection
    Dim cmdByRoyalty As ADODB.Command
    Dim rstByRoyalty As ADODB.Recordset
    Dim rstAuthors As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLAuthors As String
    ' record variables
    Dim intRoyalty As Integer
    Dim strAuthorID As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
    Cnxn.Open strCnxn

    ' Open a command object for a stored procedure
    ' with one parameter
```

```
    Dim strCnxn As String
    Dim strSQLAuthors As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
    Cnxn.Open strCnxn

    Set rstAuthors = New ADODB.Recordset
    strSQLAuthors = "SELECT au_id, au_lname, au_fname, city, phone FROM Authors"
    rstAuthors.Open strSQLAuthors, Cnxn, adOpenDynamic, adLockOptimistic, adCmdText

    'For sake of illustration, save the Recordset to a diskette in XML format
    rstAuthors.Save "a:\Pubs.xml", adPersistXML
  'clean up
    rstAuthors.Close
    Cnxn.Close
    Set rstAuthors = Nothing
    Set Cnxn = Nothing

End Sub
'EndSaveVB
```

At this point, you have arrived at your destination. You will access the Authors table as a local, disconnected Recordset. Don't forget you must have the MSPersist provider on the machine that you are using in order to access the saved file, a:\Pubs.xml.

```
'BeginSave2VB
Public Sub SaveX2()

    Dim rst As ADODB.Recordset
    Set rst = New ADODB.Recordset

    'For sake of illustration, we specify all parameters
    rst.Open "a:\Pubs.xml", "Provider=MSPersist;", adOpenForwardOnly, adLockBatchOptimistic, adCmdFile

    'Now you have a local, disconnected Recordset - Edit as you desired
    '(In this example the change makes no difference)
    rst.Find "au_lname = 'Carson'"
    If rst.EOF Then
        Debug.Print "Name not found."
        Exit Sub
    End If

    rst!city = "Chicago"
    rst.Update

    'Save changes in ADTG format this time, purely for sake of illustration.
    'Note that the previous version is still on the diskette, as a:\Pubs.xml.
    rst.Save "a:\Pubs.adtg", adPersistADTG
    rst.Close
    Set rst = Nothing

End Sub
'EndSave2VB
```

Finally, you return home. Now update the database with your changes.

```
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

    ' Open recordset using object refs to set properties
    ' that allow for updates to the database
    Set rstTitles = New ADODB.Recordset
    Set rstTitles.ActiveConnection = Cnxn
    rstTitles.CursorType = adOpenKeyset
    rstTitles.LockType = adLockOptimistic

    strSQLTitles = "titles"
    rstTitles.Open strSQLTitles

    'rstTitles.Open strSQLTitles, Cnxn, adOpenKeyset, adLockPessimistic, adCmdTable
    'the above line of code passes the same refs as the object refs listed above

    ' Change the type of the first title in the recordset
    rstTitles!Type = "database"

    ' Display the results of the change
    MsgBox "Before resync: " & vbCr & vbCr & _
        "Title - " & rstTitles!Title & vbCr & _
        "Type - " & rstTitles!Type

    ' Resync with database and redisplay results
    rstTitles.Resync
    MsgBox "After resync: " & vbCr & vbCr & _
        "Title - " & rstTitles!Title & vbCr & _
        "Type - " & rstTitles!Type

    rstTitles.CancelBatch
    rstTitles.Close

End Sub
'EndResyncVB
```

4.2.1.26Save and Open Methods Example

These three examples demonstrate how the Save and Open methods can be used together.Assume you are going on a business trip and want to take along a table from a database. Before you go, you access the data as a Recordset and save it in a transportable form. When you arrive at your destination, you access the Recordset as a local, disconnected Recordset. You make changes to the Recordset, then save it again. Finally, when you return home, you connect to the database again and update it with the changes you made on the road.
First, access and save the Authors table.

```
'BeginSaveVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub SaveX1()

    'recordset and connection variables
    Dim rstAuthors As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
```

```
    Dim strCnxn As String
    Dim strSQLAuthors As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
    Cnxn.Open strCnxn

    Set rstAuthors = New ADODB.Recordset
    strSQLAuthors = "SELECT au_id, au_lname, au_fname, city, phone FROM Authors"
    rstAuthors.Open strSQLAuthors, Cnxn, adOpenDynamic, adLockOptimistic, adCmdText

    'For sake of illustration, save the Recordset to a diskette in XML format
    rstAuthors.Save "a:\Pubs.xml", adPersistXML
  'clean up
    rstAuthors.Close
    Cnxn.Close
    Set rstAuthors = Nothing
    Set Cnxn = Nothing

End Sub
'EndSaveVB
```

At this point, you have arrived at your destination. You will access the Authors table as a local, disconnected Recordset. Don't forget you must have the MSPersist provider on the machine that you are using in order to access the saved file, a:\Pubs.xml.

```
'BeginSave2VB
Public Sub SaveX2()

    Dim rst As ADODB.Recordset
    Set rst = New ADODB.Recordset

    'For sake of illustration, we specify all parameters
    rst.Open "a:\Pubs.xml", "Provider=MSPersist;", adOpenForwardOnly, adLockBatchOptimistic, adCmdFile

    'Now you have a local, disconnected Recordset - Edit as you desired
    '(In this example the change makes no difference)
    rst.Find "au_lname = 'Carson'"
    If rst.EOF Then
        Debug.Print "Name not found."
        Exit Sub
    End If

    rst!city = "Chicago"
    rst.Update

    'Save changes in ADTG format this time, purely for sake of illustration.
    'Note that the previous version is still on the diskette, as a:\Pubs.xml.
    rst.Save "a:\Pubs.adtg", adPersistADTG
    rst.Close
    Set rst = Nothing

End Sub
'EndSave2VB
```

Finally, you return home. Now update the database with your changes.

```
'BeginSave3VB
Public Sub SaveX3()

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

    Dim Cnxn As New ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim strCnxn As String

    Set rst = New ADODB.Recordset
    ' The lock mode is batch optimistic because we are going to
    ' use the UpdateBatch method.
    rst.Open "a:\Pubs.adtg", "Provider=MSPersist;", adOpenForwardOnly,
adLockBatchOptimistic, adCmdFile

     ' Connect to the database, associate the Recordset with the connection
     ' then update the database table with the changed Recordset
    strCnxn = "Provider=SQLOLEDB;Data Source=MyServer;User
Id=sa;Password=pwd;Initial Catalog=pubs;"
    Cnxn.Open strCnxn

    rst.ActiveConnection = Cnxn
    rst.UpdateBatch

     ' clean up
    rst.Close
    Cnxn.Close
    Set rst = Nothing
    Set Cnxn = Nothing

End Sub
'EndSave3VB
```

4.2.1.27Seek Method and Index Property Example

This example uses the Recordset object's Seek method and Index property in conjunction with a given Employee ID, to locate the employee's name in the Employees table of the Nwind.mdb database.

```
'BeginSeekVB
Public Sub SeekX()

    ' To integrate this code replace the data source
    ' in the connection string

     'recordset and connection variables
    Dim rstEmployees As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLEmployees As String

    Dim strID As String
    Dim strPrompt As String
    strPrompt = "Enter an EmployeeID (e.g., 0 to 9)"

    ' Open connection
    Set Cnxn = New ADODB.Connection
```

```
    strCnxn = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
              "Data Source=c:\Program Files\Microsoft
Office\Office\Samples\northwind.mdb;" & _
              "user id=admin;password=;"
    Cnxn.Open strCnxn

     ' open recordset server-side for indexing
    Set rstEmployees = New ADODB.Recordset
    rstEmployees.CursorLocation = adUseServer
    strSQLEmployees = "employees"
    rstEmployees.Open strSQLEmployees, strCnxn, adOpenKeyset, adLockReadOnly,
adCmdTableDirect

    ' Does this provider support Seek and Index?
    If rstEmployees.Supports(adIndex) And rstEmployees.Supports(adSeek) Then
        rstEmployees.Index = "PrimaryKey"
        ' Display all the employees
            rstEmployees.MoveFirst
            Do While rstEmployees.EOF = False
                Debug.Print rstEmployees!EmployeeId; ": "; rstEmployees!firstname; " ";
_
                    rstEmployees!LastName
                rstEmployees.MoveNext
            Loop

    ' Prompt the user for an EmployeeID between 0 and 9
        rstEmployees.MoveFirst
        Do
            strID = LCase(Trim(InputBox(strPrompt, "Seek Example")))
            ' Quit if strID is a zero-length string (CANCEL, null, etc.)
            If Len(strID) = 0 Then Exit Do
            If Len(strID) = 1 And strID >= "0" And strID <= "9" Then
                rstEmployees.Seek Array(strID), adSeekFirstEQ
                If rstEmployees.EOF Then
                    Debug.Print "Employee not found."
                Else
                    Debug.Print strID; ": Employee="; rstEmployees!firstname; " "; _
                    rstEmployees!LastName; """
                End If
            End If
        Loop
    End If

     ' clean up
    rstEmployees.Close
    Set rstEmployees = Nothing
    Cnxn.Close
    Set Cnxn = Nothing

End Sub
'EndSeekVB
```

4.2.1.28Supports Method Example

This example uses the Supports method to display the options supported by a recordset opened with different cursor types. The DisplaySupport procedure is required for this procedure to run.

```
'BeginSupportsVB
```

```
        'To integrate this code
        'replace the data source and initial catalog values
        'in the connection string

Public Sub SupportsX()

        ' recordset and connection variables
    Dim rstTitles As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLTitles As String
        ' array variables
    Dim arrCursorType(4) As Integer
    Dim intIndex As Integer

    ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

    ' Fill array with CursorType constants
    arrCursorType(0) = adOpenForwardOnly
    arrCursorType(1) = adOpenKeyset
    arrCursorType(2) = adOpenDynamic
    arrCursorType(3) = adOpenStatic

    ' open recordset using each CursorType and optimistic locking
    For intIndex = 0 To 3
        Set rstTitles = New ADODB.Recordset
        rstTitles.CursorType = arrCursorType(intIndex)
        rstTitles.LockType = adLockOptimistic

        strSQLTitles = "Titles"
        rstTitles.Open strSQLTitles, Cnxn, , , adCmdTable

        Select Case arrCursorType(intIndex)
            Case adOpenForwardOnly
                Debug.Print "ForwardOnly cursor supports:"
            Case adOpenKeyset
                Debug.Print "Keyset cursor supports:"
            Case adOpenDynamic
                Debug.Print "Dynamic cursor supports:"
            Case adOpenStatic
                Debug.Print "Static cursor supports:"
        End Select

    ' call the DisplaySupport procedure from below
    ' to display the supported options
        DisplaySupport rstTitles

    Next intIndex

          ' clean up
        rstTitles.Close
        Cnxn.Close
        Set Cnxn = Nothing
        Set rstTitles = Nothing
```

```
End Sub
'EndSupportsVB


'BeginSupports2VB
Public Sub DisplaySupport(rstTemp As ADODB.Recordset)

    Dim arrConstants(11) As Long
    Dim blnSupports As Boolean
    Dim intIndex As Integer

    ' Fill array with cursor option constants
    arrConstants(0) = adAddNew
    arrConstants(1) = adApproxPosition
    arrConstants(2) = adBookmark
    arrConstants(3) = adDelete
    arrConstants(4) = adFind
    arrConstants(5) = adHoldRecords
    arrConstants(6) = adMovePrevious
    arrConstants(7) = adNotify
    arrConstants(8) = adResync
    arrConstants(9) = adUpdate
    arrConstants(10) = adUpdateBatch

    For intIndex = 0 To 10
        blnSupports = _
            rstTemp.Supports(arrConstants(intIndex))
        If blnSupports Then
            Select Case arrConstants(intIndex)
                Case adAddNew
                    Debug.Print "     AddNew"
                Case adApproxPosition
                    Debug.Print "     AbsolutePosition and AbsolutePage"
                Case adBookmark
                    Debug.Print "     blnkmark"
                Case adDelete
                    Debug.Print "     Delete"
                Case adFind
                    Debug.Print "     Find"
                Case adHoldRecords
                    Debug.Print "     Holding Records"
                Case adMovePrevious
                    Debug.Print "     MovePrevious and Move"
                Case adNotify
                    Debug.Print "     Notifications"
                Case adResync
                    Debug.Print "     Resyncing data"
                Case adUpdate
                    Debug.Print "     Update"
                Case adUpdateBatch
                    Debug.Print "     batch updating"
            End Select
        End If
    Next intIndex

End Sub
'EndSupports2VB
```

4.2.1.29Update and CancelUpdate Methods Example

This example demonstrates the Update method in conjunction with the CancelUpdate method.

```
'BeginUpdateVB
Public Sub UpdateX()

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

    ' recordset and connection variables
   Dim rstEmployees As ADODB.Recordset
   Dim Cnxn As ADODB.Connection
   Dim strCnxn As String
   Dim strSQLEmployees As String
    ' buffer variables
   Dim strOldFirst As String
   Dim strOldLast As String
   Dim strMessage As String

   ' Open connection
   Set Cnxn = New ADODB.Connection
   strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
   Cnxn.Open strCnxn

   ' Open recordset to enable changes
   Set rstEmployees = New ADODB.Recordset
   strSQLEmployees = "SELECT fname, lname FROM Employee ORDER BY lname"
   rstEmployees.Open strSQLEmployees, Cnxn, adOpenKeyset, adLockOptimistic, adCmdText

   ' Store original data
   strOldFirst = rstEmployees!fname
   strOldLast = rstEmployees!lname
   ' Change data in edit buffer
   rstEmployees!fname = "Linda"
   rstEmployees!lname = "Kobara"

   ' Show contents of buffer and get user input
   strMessage = "Edit in progress:" & vbCr & _
       "   Original data = " & strOldFirst & " " & _
       strOldLast & vbCr & "   Data in buffer = " & _
       rstEmployees!fname & " " & rstEmployees!lname & vbCr & vbCr & _
       "Use Update to replace the original data with " & _
       "the buffered data in the Recordset?"

   If MsgBox(strMessage, vbYesNo) = vbYes Then
       rstEmployees.Update
   Else
       rstEmployees.CancelUpdate
   End If

   ' show the resulting data
   MsgBox "Data in recordset = " & rstEmployees!fname & " " & _
       rstEmployees!lname
```

```
    ' restore original data because this is a demonstration
    If Not (strOldFirst = rstEmployees!fname And _
            strOldLast = rstEmployees!lname) Then
        rstEmployees!fname = strOldFirst
        rstEmployees!lname = strOldLast
        rstEmployees.Update
    End If

     ' clean up
    rstEmployees.Close
    Cnxn.Close
    Set rstEmployees = Nothing
    Set Cnxn = Nothing

End Sub
' EndUpdateVB

This example demonstrates the Update method in conjunction with the AddNew method.
' BeginUpdate2VB
Public Sub UpdateX2()

    Dim cnn1 As ADODB.Connection
    Dim rstEmployees As ADODB.Recordset
    Dim strEmpID As String
    Dim strOldFirst As String
    Dim strOldLast As String
    Dim strMessage As String
    Dim strCnn As String

    ' Open a connection.
    Set cnn1 = New ADODB.Connection
        strCnn = "Provider=sqloledb;" & _
        "Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
    cnn1.Open strCnn

    ' Open recordset with data from Employees table.
    Set rstEmployees = New ADODB.Recordset
    rstEmployees.CursorType = adOpenKeyset
    rstEmployees.LockType = adLockOptimistic
    rstEmployees.Open "employee", cnn1, , , adCmdTable

    rstEmployees.AddNew
    strEmpID = "B-S55555M"
    rstEmployees!emp_id = strEmpID
    rstEmployees!fname = "Bill"
    rstEmployees!lname = "Sornsin"

    ' Show contents of buffer and get user input.
    strMessage = "AddNew in progress:" & vbCr & _
        "Data in buffer = " & rstEmployees!emp_id & ", " & _
        rstEmployees!fname & " " & rstEmployees!lname & vbCr & vbCr & _
        "Use Update to save buffer to recordset?"

    If MsgBox(strMessage, vbYesNoCancel) = vbYes Then
        rstEmployees.Update
        ' Go to the new record and show the resulting data.
        MsgBox "Data in recordset = " & rstEmployees!emp_id & ", " & _
            rstEmployees!fname & " " & rstEmployees!lname
    Else
```

```
        rstEmployees.CancelUpdate
        MsgBox "No new record added."
    End If

    ' Delete new data because this is a demonstration.
    cnn1.Execute "DELETE FROM employee WHERE emp_id = '" & strEmpID & "'"

    rstEmployees.Close

End Sub
'EndUpdate2VB
```

4.2.1.30UpdateBatch and CancelBatch Methods Example

This example demonstrates the UpdateBatch method in conjunction with the CancelBatch method.

```
'BeginUpdateBatchVB
Public Sub UpdateBatchX()

     'To integrate this code
     'replace the data source and initial catalog values
     'in the connection string

     'connection and recordset variables
    Dim rstTitles As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLTitles As String
     'record variables
    Dim strTitle As String
    Dim strMessage As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
    Cnxn.Open strCnxn

     ' open recordset for batch uodate
    Set rstTitles = New ADODB.Recordset
    strSQLTitles = "titles"
    rstTitles.Open strSQLTitles, Cnxn, adOpenKeyset, adLockBatchOptimistic, adCmdTable

    rstTitles.MoveFirst
    ' Loop through recordset and ask user if she wants
    ' to change the type for a specified title.
    Do Until rstTitles.EOF

        If Trim(rstTitles!Type) = "psychology" Then
            strTitle = rstTitles!title
            strMessage = "Title: " & strTitle & vbCr & _
                "Change type to self help?"

            If MsgBox(strMessage, vbYesNo) = vbYes Then
                rstTitles!Type = "self_help"
            End If
        End If
```

```
    rstTitles.MoveNext
    Loop

    ' Ask the user if she wants to commit to all the
    ' changes made above.
    If MsgBox("Save all changes?", vbYesNo) = vbYes Then
        rstTitles.UpdateBatch
    Else
        rstTitles.CancelBatch
    End If

    ' Print current data in recordset.
    rstTitles.Requery
    rstTitles.MoveFirst
    Do While Not rstTitles.EOF
        Debug.Print rstTitles!title & " - " & rstTitles!Type
        rstTitles.MoveNext
    Loop

    ' Restore original values because this is a demonstration.
    rstTitles.MoveFirst
    Do Until rstTitles.EOF
        If Trim(rstTitles!Type) = "self_help" Then
            rstTitles!Type = "psychology"
        End If
        rstTitles.MoveNext
    Loop
    rstTitles.UpdateBatch

    rstTitles.Close

End Sub
'EndUpdateBatchVB
```

4.2.2 Properties

- AbsolutePage, PageCount, and PageSize Properties Example
- AbsolutePosition and CursorLocation Properties Example
- ActiveCommand Property Example
- ActiveConnection, CommandText, CommandTimeout, CommandType, Size, and Direction Properties Example
- ActualSize and DefinedSize Properties Example
- Attributes and Name Properties Example
- BOF, EOF, and Bookmark Properties Example
- CacheSize Property Example
- ConnectionString, ConnectionTimeout, and State Properties Example
- Count Property Example
- CursorType, LockType, and EditMode Properties Example
- Description, HelpContext, HelpFile, NativeError, Number, Source, and SQLState Properties Example
- EOS and LineSeparator Properties, SkipLine Method Example
- Filter and RecordCount Properties Example
- IsolationLevel and Mode Properties Example
- Item Property Example
- MarshalOptions Property Example
- MaxRecords Property Example

- NumericScale and Precision Properties Example
- Optimize Property Example
- OriginalValue and UnderlyingValue Properties Example
- Prepared Property Example
- Provider and DefaultDatabase Properties Example
- Sort Property Example
- Source Property Example
- State Property Example
- Status Property Example(Recordset)
- StayInSync Property Example
- Type Property Example (Field)
- Type Property Example (Property)
- Value Property Example
- Version Property Example

4.2.2.1AbsolutePage, PageCount, and PageSize Properties Example

```
'BeginAbsolutePageVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub AbsolutePageX()

    'recordset and connection variables
Dim rstEmployees As ADODB.Recordset
Dim Cnxn As ADODB.Connection
Dim strCnxn As String
Dim strSQL As String
    'record variables
Dim strMessage As String
Dim intPage As Integer
Dim intPageCount As Integer
Dim intRecord As Integer

    'Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=;"
    Cnxn.Open strCnxn

  ' Open employee recordset
  ' Use client cursor to enable AbsolutePosition property
  Set rstEmployees = New ADODB.Recordset
  strSQL = "employee"
  rstEmployees.Open strSQL, strCnxn, adUseClient, adLockReadOnly, adCmdTable

  ' Display names and hire dates, five records at a time
  rstEmployees.PageSize = 5
  intPageCount = rstEmployees.PageCount
  For intPage = 1 To intPageCount
      rstEmployees.AbsolutePage = intPage
      strMessage = ""
      For intRecord = 1 To rstEmployees.PageSize
          strMessage = strMessage & _
              rstEmployees!fname & " " & _
              rstEmployees!lname & " " & _
```

```
                rstEmployees!hire_date & vbCr
            rstEmployees.MoveNext
            If rstEmployees.EOF Then Exit For
        Next intRecord
        MsgBox strMessage
    Next intPage

     ' clean up
    rstEmployees.Close
    Cnxn.Close
    Set rstEmployees = Nothing
    Set Cnxn = Nothing
End Sub
'EndAbsolutePageVB
```

4.2.2.2AbsolutePosition and CursorLocation Properties

Example

This example demonstrates how the AbsolutePosition property can track the progress of a loop that enumerates all the records of a Recordset. It uses the CursorLocation property to enable the AbsolutePosition property by setting the cursor to a client cursor.

```
'BeginAbsolutePositionVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub AbsolutePositionX()

    'recordset and connection variables
Dim rstEmployees As ADODB.Recordset
Dim Cnxn As ADODB.Connection
Dim strCnxn As String
Dim strSQL As String
    'record variables
Dim strMessage As String

    'Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=;"
    Cnxn.Open strCnxn

    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    ' Open Employee recordset with
    ' Client-side cursor to enable AbsolutePosition property
    Set rstEmployees = New ADODB.Recordset
    strSQL = "employee"
    rstEmployees.Open strSQL, strCnxn, adUseClient, adLockReadOnly, adCmdTable

    ' Enumerate Recordset
    Do While Not rstEmployees.EOF
        ' Display current record information
        strMessage = "Employee: " & rstEmployees!lname & vbCr & _
            "(record " & rstEmployees.AbsolutePosition & _
            " of " & rstEmployees.RecordCount & ")"
        If MsgBox(strMessage, vbOKCancel) = vbCancel Then Exit Do
        rstEmployees.MoveNext
```

```
    Loop

     ' clean up
    rstEmployees.Close
    Cnxn.Close
    Set rstEmployees = Nothing
    Set Cnxn = Nothing

End Sub
'EndAbsolutePositionVB
```

4.2.2.3Active Command Property Example

This example demonstrates the ActiveCommand property.
A subroutine is given a Recordset object whose ActiveCommand property is used to display the command text and parameter that created the Recordset.

```
'BeginActiveCommandVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub ActiveCommandX()

        'recordset and connection variables
    Dim cmd As ADODB.Command
    Dim rst As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
        'record variables
    Dim strPrompt As String
    Dim strName As String

    Set Cnxn = New ADODB.Connection
    Set cmd = New ADODB.Command

    strPrompt = "Enter an author's name (e.g., Ringer): "
    strName = Trim(InputBox(strPrompt, "ActiveCommandX Example"))
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "

        'create SQL command string
    cmd.CommandText = "SELECT * FROM Authors WHERE au_lname = ?"
    cmd.Parameters.Append cmd.CreateParameter("LastName", adChar, adParamInput, 20, strName)

    Cnxn.Open strCnxn
    cmd.ActiveConnection = Cnxn

        'create the recordset by executing command string
    Set rst = cmd.Execute(, , adCmdText)
        'see the results
    Call ActiveCommandXprint(rst)

     ' clean up
    rst.Close
    Cnxn.Close
    Set rst = Nothing
    Set Cnxn = Nothing
```

```
End Sub
'EndActiveCommandVB
```

The **ActiveCommandXprint** routine is given only a **Recordset** object, yet it must print the **command** text and parameter that created the **Recordset**. This can be done because the **Recordset** object's **ActiveCommand** property yields the associated **Command** object. The **Command** object's **CommandText** property yields the parameterized command that created the **Recordset**. The **Command** object's Parameters collection yields the value that was substituted for the command's parameter placeholder ("?").
Finally, an error message or the author's name and ID are printed.

```
'BeginActiveCommandPrintVB
Public Sub ActiveCommandXprint(rstp As ADODB.Recordset)

    Dim strName As String

    strName = rstp.ActiveCommand.Parameters.Item("LastName").Value

    Debug.Print "Command text = "; rstp.ActiveCommand.CommandText; """
    Debug.Print "Parameter = "; strName; """

    If rstp.BOF = True Then
        Debug.Print "Name = "; strName; ", not found."
    Else
        Debug.Print "Name = "; rstp!au_fname; " "; rstp!au_lname; _
                ", author ID = "; rstp!au_id; """
    End If

    rstp.Close
    Set rstp = Nothing

End Sub
'EndActiveCommandPrintVB
```

4.2.2.4ActiveConnection, CommandText, CommandTimeout, CommandType, Size, and Direction Properties Example

This example uses the ActiveConnection, CommandText, CommandTimeout, CommandType, Size, and Direction properties to execute a stored procedure.

```
'BeginActiveConnectionVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub ActiveConnectionX()

     'recordset, command and connection variables
    Dim Cnxn As ADODB.Connection
    Dim cmdByRoyalty As ADODB.Command
    Dim prmByRoyalty As ADODB.Parameter
    Dim rstByRoyalty As ADODB.Recordset
    Dim rstAuthors As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLAuthors As String
    Dim strSQLByRoyalty As String
     'record variables
    Dim intRoyalty As Integer
    Dim strAuthorID As String
```

```
    ' Define a command object for a stored procedure
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

    Set cmdByRoyalty = New ADODB.Command
    Set cmdByRoyalty.ActiveConnection = Cnxn
    ' Set the criteria
    strSQLByRoyalty = "byroyalty"
    cmdByRoyalty.CommandText = strSQLByRoyalty
    cmdByRoyalty.CommandType = adCmdStoredProc
    cmdByRoyalty.CommandTimeout = 15

    ' Define the stored procedure's input parameter
    intRoyalty = Trim(InputBox("Enter royalty:"))
    Set prmByRoyalty = New ADODB.Parameter
    prmByRoyalty.Type = adInteger
    prmByRoyalty.Size = 3
    prmByRoyalty.Direction = adParamInput
    prmByRoyalty.Value = intRoyalty

    cmdByRoyalty.Parameters.Append prmByRoyalty

    ' Create a recordset by executing the command.
    Set rstByRoyalty = cmdByRoyalty.Execute()

    ' Open the Authors Table to get author names for display
    Set rstAuthors = New ADODB.Recordset
    strSQLAuthors = "Authors"

    'rstAuthors.Open strSQLAuthors, strCnxn, , , adCmdTable
    rstAuthors.Open SQLAuthors, strCnx, adOpenForwardOnly, adLockReadOnly,
adCmdTable
    'the above two lines of code are identical as the default values for
    'CursorType and LockType arguments match those shown

    ' Print the recordset and add author names from Table
    Debug.Print "Authors with " & intRoyalty & _
        " percent royalty"

     Do Until rstByRoyalty.EOF
        strAuthorID = rstByRoyalty!au_id
        Debug.Print , rstByRoyalty!au_id & ", ";
        rstAuthors.Filter = "au_id = '" & strAuthorID & "'"
        Debug.Print rstAuthors!au_fname & " " & _
            rstAuthors!au_lname
        rstByRoyalty.MoveNext
     Loop

    rstByRoyalty.Close
    rstAuthors.Close
    Cnxn.Close

     ' clean up
    Set Cnxn = Nothing
    Set rstAuthors = Nothing
    Set rstByRoyalty = Nothing
```

```
End Sub
'EndActiveConnectionVB
```

4.2.2.5ActualSize and DefinedSize Properties Example

This example uses the ActualSize and DefinedSize properties to display the defined size and actual size of a field.

```
'BeginActualSizeVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub ActualSizeX()

    'recordset and connection variables
    Dim rstStores As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim SQLStores As String
    Dim strCnxn As String
    'record variables
    Dim strMessage As String

    ' Open a recordset for the Stores table
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Northwind;User Id=sa;Password=; "
    Set rstStores = New ADODB.Recordset

    SQLStores = "Suppliers"
    'rstStores.Open SQLStores, strCnx, , , adCmdTable
    rstStores.Open SQLStores, strCnx, adOpenForwardOnly, adLockReadOnly, adCmdTable
    'the above two lines of code are identical as the default values for
    'CursorType and LockType arguments match those indicated

    ' Loop through the recordset displaying the contents
    ' of the store_name field, the field's defined size,
    ' and its actual size.
    rstStores.MoveFirst

    Do Until rstStores.EOF
        strMessage = "Company name: " & rstStores!CompanyName & _
        vbCrLf & "Defined size: " & _
        rstStores!CompanyName.DefinedSize & _
        vbCrLf & "Actual size: " & _
        rstStores!CompanyName.ActualSize & vbCrLf

        MsgBox strMessage, vbOKCancel, "ADO ActualSize Property (Visual Basic)"
        rstStores.MoveNext
    Loop

    ' clean up
    rstStores.Close
    Cnxn.Close
    Set rstStores = Nothing
    Set Cnxn = Nothing
```

```
End Sub
'EndActualSizeVB
```

4.2.2.6Attributes and Name Properties Example

This example displays the value of the Attributes property for Connection, Field, and Property objects. It uses the Name property to display the name of each Field and Property object.

```
'BeginAttributesVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub AttributesX()

    'recordset and connection variables
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim rstEmployee As ADODB.Recordset
    Dim strSQLEmployee As String
    'record variables
    Dim adoField As ADODB.Field
    Dim adoProp As ADODB.Property

    ' Open connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
    Set Cnxn = New ADODB.Connection
    Cnxn.Open strCnxn

    ' Open recordset
    Set rstEmployee = New ADODB.Recordset
    strSQLEmployee = "employee"
    'rstEmployee.Open strSQLEmployee, Cnxn, , , adCmdTable
    rstEmployee.Open strSQLEmployee, Cnxn, adOpenForwardOnly, adLockReadOnly, adCmdTable
    'the above two lines openign the recordset are identical as
    'the default values for CursorType and LockType arguments match those shown

    ' Display the attributes of the connection
    Debug.Print "Connection attributes = " & Cnxn.Attributes

    ' Display the property attributes of the Employee Table
    Debug.Print "Property attributes:"
    For Each adoProp In rstEmployee.Properties
        Debug.Print "     " & adoProp.Name & " = " & adoProp.Attributes
    Next adoProp

    ' Display the field attributes of the Employee Table
    Debug.Print "Field attributes:"
    For Each adoField In rstEmployee.Fields
        Debug.Print "     " & adoField.Name & " = " & adoField.Attributes
    Next adoField

    ' Display fields of the Employee Table which are NULLABLE
    Debug.Print "NULLABLE Fields:"
    For Each adoField In rstEmployee.Fields
        If CBool(adoField.Attributes And adFldIsNullable) Then
```

```
            Debug.Print "     " & adoField.Name
        End If
    Next adoField

     ' clean up
    rstEmployee.Close
    Cnxn.Close
    Set rstEmployee = Nothing
    Set Cnxn = Nothing

End Sub
'EndAttributesVB
```

4.2.2.7BOF, EOF, and Bookmark Properties Example

This example uses the BOF and EOF properties to display a message if a user tries to move past the first or last record of a Recordset. It uses the Bookmark property to let the user flag a record in a Recordset and return to it later.

```
'BeginBOFVB

     'To integrate this code
     'replace the data source and initial catalog values
     'in the connection string

Public Sub BOFX()

     'recordset and connection variables
    Dim Cnxn As ADODB.Connection
    Dim rstPublishers As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLPubs As String
     'record variables
    Dim strMessage As String
    Dim intCommand As Integer
    Dim varBookmark As Variant

     ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

     ' Open recordset and use client cursor
     ' to enable AbsolutePosition property
    Set rstPublishers = New ADODB.Recordset
    strSQLPubs = "SELECT pub_id, pub_name FROM publishers ORDER BY pub_name"
    rstPublishers.Open strSQLPubs, strCnxn, adUseClient, adOpenStatic, adCmdText

    rstPublishers.MoveFirst
    Do Until rstPublishers.EOF
        ' Display information about current record
        ' and get user input
        strMessage = "Publisher: " & rstPublishers!pub_name & _
            vbCr & "(record " & rstPublishers.AbsolutePosition & _
            " of " & rstPublishers.RecordCount & ")" & vbCr & vbCr & _
            "Enter command:" & vbCr & _
            "[1 - next / 2 - previous /" & vbCr & _
            "3 - set bookmark / 4 - go to bookmark]"
        intCommand = Val(InputBox(strMessage))
```

```
        ' Check user input
        Select Case intCommand
            Case 1
            ' Move forward trapping for EOF
                rstPublishers.MoveNext
                If rstPublishers.EOF Then
                    MsgBox "Moving past the last record." & _
                        vbCr & "Try again."
                    rstPublishers.MoveLast
                End If

            Case 2
            ' Move backward trapping for BOF
                rstPublishers.MovePrevious
                If rstPublishers.BOF Then
                    MsgBox "Moving past the first record." & _
                        vbCr & "Try again."
                    rstPublishers.MoveFirst
                End If

            Case 3
            ' Store the bookmark of the current record
                varBookmark = rstPublishers.Bookmark

            Case 4
            ' Go to the record indicated by the stored bookmark
                If IsEmpty(varBookmark) Then
                    MsgBox "No Bookmark set!"
                Else
                    rstPublishers.Bookmark = varBookmark
                End If

            Case Else
                Exit Do
        End Select
    Loop

     ' clean up
    Cnxn.Close
    rstPublishers.Close
    Set Cnxn = Nothing
    Set rstPublishers = Nothing

End Sub
'EndBOFVB
```

This example uses the Bookmark and Filter properties to create a limited view of the Recordset. Only records referenced by the array of bookmarks are accessible.

```
'BeginBOF2VB
Public Sub BOFX2()

    'recordset and connection variables
    Dim rs As New ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strSQL As String
    Dim strCnxn As String

    Dim bmk(10)
```

```
        'open the recordset client-side
    Set rs = New ADODB.Recordset
    strSQL = "Select * from Authors"
    rs.Open strSQL, Cnxn, adUseClient, adLockReadOnly, adCmdText
    Debug.Print "Number of records before filtering: ", rs.RecordCount

    Dim ii As Integer
    ii = 0

    If rs.EOF <> True And ii < 11 Then
        Do
            bmk(ii) = rs.Bookmark
            ii = ii + 1
        rs.Move 2
        Loop Until rs.EOF
    End If

    rs.Filter = bmk
    Debug.Print "Number of records after filtering: ", rs.RecordCount

    rs.MoveFirst
    If rs.EOF <> True Then
        Do
            Debug.Print rs.AbsolutePosition, rs("au_lname")
            rs.MoveNext
        Loop Until rs.EOF
    End If

     ' clean up
    rs.Close
    Cnxn.Close
    Set rs = Nothing
    Set Cnxn = Nothing

End Sub
'EndBOF2VB
```

4.2.2.8CacheSize Property Example

This example uses the CacheSize property to show the difference in performance for an operation performed with and without a 30-record cache.

```
'BeginCacheSizeVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub CacheSizeX()

     'recordset and connection variables
    Dim rstRoySched As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strSQLSched As String
    Dim strCnxn As String
     'record variables
    Dim sngStart As Single
    Dim sngEnd As Single
    Dim sngNoCache As Single
```

```
    Dim sngCache As Single
    Dim intLoop As Integer
    Dim strTemp As String

    ' Open the connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "

    ' Open the RoySched Table
    Set rstRoySched = New ADODB.Recordset
    strSQLSched = "roysched"
    rstRoySched.Open strSQLSched, strCnxn, , , adCmdTable

    ' Enumerate the Recordset object twice and
    ' record the elapsed time
    sngStart = Timer

    For intLoop = 1 To 2

        rstRoySched.MoveFirst

        If Not rstRoySched.EOF Then
            ' Execute a simple operation for the
            ' performance test
          Do
                strTemp = rstRoySched!title_id
          rstRoySched.MoveNext
          Loop Until rstRoySched.EOF
        End If

    Next intLoop

    sngEnd = Timer
    sngNoCache = sngEnd - sngStart

    ' Cache records in groups of 30 records.
    rstRoySched.MoveFirst
    rstRoySched.CacheSize = 30
    sngStart = Timer

    ' Enumerate the Recordset object twice and record
    ' the elapsed time
    For intLoop = 1 To 2

        rstRoySched.MoveFirst
        Do While Not rstRoySched.EOF
            ' Execute a simple operation for the
            ' performance test
            strTemp = rstRoySched!title_id
        rstRoySched.MoveNext
        Loop

    Next intLoop

    sngEnd = Timer
    sngCache = sngEnd - sngStart

    ' Display performance results.
```

```
    MsgBox "Caching Performance Results:" & vbCr & _
        "     No cache: " & Format(sngNoCache, "##0.000") & " seconds" & vbCr & _
        "     30-record cache: " & Format(sngCache, "##0.000") & " seconds"

    ' clean up
    Cnxn.Close
    rstRoySched.Close
    Set Cnxn = Nothing
    Set rstRoySched = Nothing

End Sub
'EndCacheSizeVB
```

4.2.2.9ConnectionString, ConnectionTimeout, and State Properties Example

This example demonstrates different ways of using the ConnectionString property to open a Connection object. It also uses the ConnectionTimeout property to set a connection timeout period, and the State property to check the state of the connections. The GetState function is required for this procedure to run.

```
'BeginConnectionStringVB

    'To integrate this code replace
    'the database, DSN or Data Source values

Public Sub ConnectionStringX()

    Dim Cnxn1 As ADODB.Connection
    Dim Cnxn2 As ADODB.Connection
    Dim Cnxn3 As ADODB.Connection
    Dim Cnxn4 As ADODB.Connection

    ' Open a connection without using a Data Source Name (DSN)
    Set Cnxn1 = New ADODB.Connection
    Cnxn1.ConnectionString = "driver={SQL
Server};server=srv;uid=sa;pwd=pwd;database=Pubs"
    Cnxn1.ConnectionTimeout = 30
    Cnxn1.Open

    ' Open a connection using a DSN and ODBC tags
    Set Cnxn2 = New ADODB.Connection
    Cnxn2.ConnectionString = "Provider=sqloledb;Data Source=MyServer;Initial
Catalog=Pubs;User Id=sa;Password=;"
    Cnxn2.Open

    ' Open a connection using a DSN and OLE DB tags
    Set Cnxn3 = New ADODB.Connection
    Cnxn3.ConnectionString = "Data Source=Pubs;User ID=sa;Password=pwd;"
    Cnxn3.Open

    ' Open a connection using a DSN and individual
    ' arguments instead of a connection string
    Set Cnxn4 = New ADODB.Connection
    Cnxn4.Open "Pubs", "sa", "pwd"

    ' Display the state of the connections using
    ' GetState function from below
    MsgBox "Cnxn1 state: " & GetState(Cnxn1.State) & vbCr & _
        "Cnxn2 state: " & GetState(Cnxn2.State) & vbCr & _
```

```
        "Cnxn3 state: " & GetState(Cnxn3.State) & vbCr & _
        "Cnxn4 state: " & GetState(Cnxn4.State)

    Cnxn4.Close
    Cnxn3.Close
    Cnxn2.Close
    Cnxn1.Close

End Sub

Public Function GetState(intState As Integer) As String

    Select Case intState
        Case adStateClosed
            GetState = "adStateClosed"
        Case adStateOpen
            GetState = "adStateOpen"
    End Select

End Function
'EndConnectionStringVB
```

4.2.2.10Count Property Example

This example demonstrates the Count property with two collections in the Employee database. The property obtains the number of objects in each collection, and sets the upper limit for loops that enumerate these collections. Another way to enumerate these collections without using the Count property would be to use For Each...Next statements.

```
'BeginCountVB

     'To integrate this code
     'replace the data source and initial catalog values
     'in the connection string

Public Sub CountX()

     ' recordset and connection variables
    Dim rstEmployees As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strSQLEmployees As String
    Dim strCnxn As String

    Dim intLoop As Integer

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

    ' Open recordset with data from Employee table
    Set rstEmployees = New ADODB.Recordset
    strSQLEmployees = "Employee"
    'rstEmployees.Open strSQLEmployee, Cnxn, , , adCmdTable
    rstEmployees.Open strSQLEmployees, Cnxn, adOpenForwardOnly, adLockReadOnly,
adCmdTable
    'the above two lines opening the recordset are identical as
    'the default values for CursorType and LockType arguments match those specified
```

```
    ' Print information about Fields collection
    Debug.Print rstEmployees.Fields.Count & " Fields in Employee"

    For intLoop = 0 To rstEmployees.Fields.Count - 1
        Debug.Print "     " & rstEmployees.Fields(intLoop).Name
    Next intLoop

    ' Print information about Properties collection
    Debug.Print rstEmployees.Properties.Count & " Properties in Employee"

    For intLoop = 0 To rstEmployees.Properties.Count - 1
        Debug.Print "     " & rstEmployees.Properties(intLoop).Name
    Next intLoop

     ' clean up
    rstEmployees.Close
    Cnxn.Close
    Set rstEmployees = Nothing
    Set Cnxn = Nothing

End Sub
'EndCountVB
```

4.2.2.11CursorType, LockType, and EditMode Properties Example

This example demonstrates setting the CursorType and LockType properties before opening a Recordset. It also shows the value of the EditMode property under various conditions. The EditModeOutput function is required for this procedure to run.

```
'BeginEditModeVB

     'To integrate this code
     'replace the data source and initial catalog values
     'in the connection string

Public Sub EditModeX()

     ' recordset variables
    Dim rstEmployees As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim SQLEmployees As String

     ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
    Cnxn.Open strCnxn

       ' set recordset properties through object refs
       ' instead of through arguments to Open method
    Set rstEmployees = New ADODB.Recordset
    Set rstEmployees.ActiveConnection = Cnxn
    rstEmployees.CursorLocation = adUseClient
    rstEmployees.CursorType = adOpenStatic
    rstEmployees.LockType = adLockBatchOptimistic

     ' open recordset with data from Employee table
    SQLEmployees = "employee"
```

```
    rstEmployees.Open SQLEmployees, , , , adCmdTable

    ' Show the EditMode property under different editing states
    rstEmployees.AddNew
        rstEmployees!emp_id = "T-T55555M"
        rstEmployees!fname = "temp_fname"
        rstEmployees!lname = "temp_lname"
            'call function below
            'to output results to debug window
        EditModeOutput "After AddNew:", rstEmployees.EditMode
        rstEmployees.UpdateBatch
        EditModeOutput "After UpdateBatch:", rstEmployees.EditMode
        rstEmployees!fname = "test"
        EditModeOutput "After Edit:", rstEmployees.EditMode
    rstEmployees.Close

    ' Delete new record because this is a demonstration
    Cnxn.Execute "DELETE FROM employee WHERE emp_id = 'T-T55555M'"

      Cnxn.Close
      Set rstEmployees = Nothing
      Set Cnxn = Nothing

End Sub

Public Function EditModeOutput(strTemp As String, _
    intEditMode As Integer)

    ' Print report based on the value of the EditMode
    ' property
    Debug.Print strTemp
    Debug.Print "    EditMode = ";

    Select Case intEditMode
        Case adEditNone
            Debug.Print "adEditNone"
        Case adEditInProgress
            Debug.Print "adEditInProgress"
        Case adEditAdd
            Debug.Print "adEditAdd"
    End Select

End Function
'EndEditModeVB
```

4.2.2.12Description, HelpContext, HelpFile, NativeError, Number, Source, and SQLState Properties Example

This example triggers an error, traps it, and displays the Description, HelpContext, HelpFile, NativeError, Number, Source, and SQLState properties of the resulting Error object.

```
'BeginDescriptionVB
Public Sub DescriptionX()

    Dim Cnxn As ADODB.Connection
    Dim Err As ADODB.Error
    Dim strError As String

    On Error GoTo ErrorHandler
```

```
    ' Intentionally trigger an error
    Set Cnxn = New ADODB.Connection
    Cnxn.Open "nothing"

    Exit Sub

ErrorHandler:

    ' Enumerate Errors collection and display
    ' properties of each Error object
    For Each Err In Cnxn.Errors

        strError = "Error #" & Err.Number & vbCr & _
            "     " & Err.Description & vbCr & _
            "     (Source: " & Err.Source & ")" & vbCr & _
            "     (SQL State: " & Err.SQLState & ")" & vbCr & _
            "     (NativeError: " & Err.NativeError & ")" & vbCr
        If Err.HelpFile = "" Then
            strError = strError & "     No Help file available"
        Else
            strError = strError & _
                "     (HelpFile: " & Err.HelpFile & ")" & vbCr & _
                "     (HelpContext: " & Err.HelpContext & ")" & _
                vbCr & vbCr
        End If

    Debug.Print strError
    Next

    Resume Next

End Sub
'EndDescriptionVB
```

4.2.2.13EOS and LineSeparator Properties and SkipLine Method Example

This example demonstrates how to manipulate text streams one line at a time. The effect of changing the line separator from the default carriage return/linefeed (adCRLF) to simply linefeed (adLF) or carriage return (adCR) is shown.

```
'BeginSkipLineVB
Public Sub SkipLineX()
    'Declare variables
    Dim i As Integer
    Dim objStream As Stream
    Dim strLine, strChar As String

    'Instantiate and open stream
    Set objStream = New Stream
    objStream.Open

    'Set line separator to line feed
    objStream.LineSeparator = adLF

    'Load text content of list box into stream
```

```
    'One line at a time
    For i = 0 To (List1.ListCount - 1)
        objStream.WriteText List1.List(i), adWriteLine
    Next

    'Display the entire stream
    Debug.Print "Whole Stream:"
    objStream.Position = 0
    Debug.Print objStream.ReadText

    'Display the first line
    Debug.Print "First Line:"
    objStream.Position = 0
    strLine = objStream.ReadText(adReadLine)
    Debug.Print strLine
    Debug.Print "Line length: " + Str(Len(strLine))

    'Skip a line, then display another line
    Debug.Print "Third Line:"
    objStream.SkipLine
    strLine = objStream.ReadText(adReadLine)
    Debug.Print strLine
    Debug.Print "Line length: " + Str(Len(strLine))

    'Switch line separator to carriage return
    'All items from list will be considered one line
    'Assuming no CRs have been loaded into stream
    Debug.Print "Whole Stream/First Line:"
    objStream.Position = 0
    objStream.LineSeparator = adCR
    strLine = objStream.ReadText(adReadLine)
    Debug.Print strLine
    Debug.Print "Line length: " + Str(Len(strLine))
    Debug.Print "Stream size: " + Str(objStream.Size)

    'Use EOS to Determine End of Stream
    Debug.Print "Character by character:"
    objStream.Position = 0
    Do Until objStream.EOS
        strChar = objStream.ReadText(1)
        Debug.Print strChar
    Loop
End Sub
'EndSkipLineVB
```

4.2.2.14Filter and RecordCount Properties Example

This example opens a Recordset on the Publishers table in the Pubs database. It then uses the Filter property to limit the number of visible records to those publishers in a particular country or region. The RecordCount property is used to show the difference between the filtered and unfiltered recordsets.

```
'BeginFilterVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub FilterX()
```

```
    ' recordset variables
    Dim rstPublishers As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim SQLPublishers As String

    ' criteria variables
    Dim intPublisherCount As Integer
    Dim strCountry As String
    Dim strMessage As String

    ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

    ' open recordset with data from Publishers table
    Set rstPublishers = New ADODB.Recordset
    SQLPublishers = "publishers"
    rstPublishers.Open SQLPublishers, strCnxn, adOpenStatic, , adCmdTable

    intPublisherCount = rstPublishers.RecordCount

    ' get user input
    strCountry = Trim(InputBox("Enter a country to filter on (e.g. USA):"))

    If strCountry <> "" Then
        ' open a filtered Recordset object
        rstPublishers.Filter = "Country ='" & strCountry & "'"

        If rstPublishers.RecordCount = 0 Then
            MsgBox "No publishers from that country."
        Else
            ' print number of records for the original recordset
            ' and the filtered recordset
            strMessage = "Orders in original recordset: " & _
                vbCr & intPublisherCount & vbCr & _
                "Orders in filtered recordset (Country = '" & _
                strCountry & "'): " & vbCr & _
                rstPublishers.RecordCount
            MsgBox strMessage
        End If

    End If

    ' clean up
    rstPublishers.Close
    Cnxn.Close
    Set rstPublishers = Nothing
    Set Cnxn = Nothing

End Sub

'EndFilterVB
```

Note When you know the data you want to select, it's usually more efficient to open a Recordset with an SQL statement. This example shows how you can create just one

Recordset and obtain records from a particular country.
'BeginFilter2VB

```
    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub FilterX2()

    Dim rstPublishers As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strSQLPublishers As String
    Dim strCnxn As String

     ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
    Cnxn.Open strCnxn

    ' open recordset with criteria from Publishers table
    Set rstPublishers = New ADODB.Recordset
    strSQLPublishers = "SELECT * FROM publishers WHERE Country = 'USA'"
    rstPublishers.Open strSQLPublishers, Cnxn, adOpenStatic, adLockReadOnly, adCmdText

    ' print recordset
    rstPublishers.MoveFirst
    Do While Not rstPublishers.EOF
        Debug.Print rstPublishers!pub_name & ", " & rstPublishers!country
        rstPublishers.MoveNext
    Loop

     ' clean up
    rstPublishers.Close
    Cnxn.Close
    Set rstPublishers = Nothing
    Set Cnxn = Nothing

End Sub
```

'EndFilter2VB

4.2.2.15IsolationLevel and Mode Properties Example

This example uses the Mode property to open an exclusive connection, and the IsolationLevel property to open a transaction that is conducted in isolation of other transactions.
'BeginIsolationLevelVB

```
    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub IsolationLevelX()

    Dim Cnxn As ADODB.Connection
    Dim rstTitles As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLTitles As String
```

```
    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Mode = adModeShareExclusive
    Cnxn.IsolationLevel = adXactIsolated
    Cnxn.Open strCnxn

     ' open Titles table
    Set rstTitles = New ADODB.Recordset
    strSQLTitles = "titles"
    rstTitles.Open strSQLTitles, Cnxn, adOpenDynamic, adLockPessimistic, adCmdTable

    Cnxn.BeginTrans

    ' Display connection mode
    If Cnxn.Mode = adModeShareExclusive Then
        MsgBox "Connection mode is exclusive."
    Else
        MsgBox "Connection mode is not exclusive."
    End If

    ' Display isolation level
    If Cnxn.IsolationLevel = adXactIsolated Then
        MsgBox "Transaction is isolated."
    Else
        MsgBox "Transaction is not isolated."
    End If

    ' Change the type of psychology titles
    Do Until rstTitles.EOF
        If Trim(rstTitles!Type) = "psychology" Then
            rstTitles!Type = "self_help"
            rstTitles.Update
        End If
        rstTitles.MoveNext
    Loop

    ' Print current data in recordset
    rstTitles.Requery
    Do While Not rstTitles.EOF
        Debug.Print rstTitles!Title & " - " & rstTitles!Type
        rstTitles.MoveNext
    Loop

    ' Restore original data
    Cnxn.RollbackTrans
    rstTitles.Close
    Cnxn.Close
    Set rstTitles = Nothing
    Set Cnxn = Nothing

End Sub
'EndIsolationLevelVB
```

4.2.2.16Item Property Example

This example demonstrates how the Item property accesses members of a collection. The

example opens the Authors table of the Pubs database with a parameterized command. The parameter in the command issued against the database is accessed from the Command object's Parameters collection by index and name. The fields of the returned Recordset are then accessed from that object's Fields collection by index and name.

```
'BeginItemVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub ItemX()

    Dim Cnxn As ADODB.Connection
    Dim rstAuthors As ADODB.Recordset
    Dim cmd As ADODB.Command
    Dim prm As ADODB.Parameter
    Dim fld As ADODB.Field
    Dim strCnxn As String

    Dim ix As Integer
    Dim limit As Long
    Dim Column(0 To 8) As Variant

    Set Cnxn = New ADODB.Connection
    Set rstAuthors = New ADODB.Recordset
    Set cmd = New ADODB.Command

    'Set the array with the Authors table field (column) names
    Column(0) = "au_id"
    Column(1) = "au_lname"
    Column(2) = "au_fname"
    Column(3) = "phone"
    Column(4) = "address"
    Column(5) = "city"
    Column(6) = "state"
    Column(7) = "zip"
    Column(8) = "contract"

    cmd.CommandText = "SELECT * FROM Authors WHERE state = ?"
    Set prm = cmd.CreateParameter("ItemXparm", adChar, adParamInput, 2, "CA")
    cmd.Parameters.Append prm
     ' set connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn
    cmd.ActiveConnection = Cnxn
     ' open recordset
    rstAuthors.Open cmd, , adOpenStatic, adLockReadOnly
    'Connection and CommandType are omitted because
    'a Command object is provided

    Debug.Print "The Parameters collection accessed by index..."
    Set prm = cmd.Parameters.Item(0)
    Debug.Print "Parameter name = "; prm.Name; "", value = ""; prm.Value; """
    Debug.Print

    Debug.Print "The Parameters collection accessed by name..."
    Set prm = cmd.Parameters.Item("ItemXparm")
    Debug.Print "Parameter name = "; prm.Name; "", value = ""; prm.Value; """
```

```
        Debug.Print

        Debug.Print "The Fields collection accessed by index..."

        rstAuthors.MoveFirst
        limit = rstAuthors.Fields.count - 1
        For ix = 0 To limit
            Set fld = rstAuthors.Fields.Item(ix)
            Debug.Print "Field "; ix; ": Name = "; fld.Name; _
                        ", Value = "; fld.Value; """
        Next ix

        Debug.Print

        Debug.Print "The Fields collection accessed by name..."

        rstAuthors.MoveFirst
        For ix = 0 To 8
            Set fld = rstAuthors.Fields.Item(Column(ix))
            Debug.Print "Field name = "; fld.Name; ", Value = "; fld.Value; """
        Next ix

        rstAuthors.Close
        Cnxn.Close
        Set rstAuthors = Nothing
        Set Cnxn = Nothing
        Set cmd = Nothing

End Sub
'EndItemVB
```

4.2.2.17MarshalOptions Property Example

This example uses the MarshalOptions property to specify what rows are sent back to the server—All Rows or only Modified Rows.

```
'BeginMarshalOptionsVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub MarshalOptionsX()

    Dim rstEmployees As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLEmployees As String

    Dim strOldFirst As String
    Dim strOldLast As String
    Dim strMessage As String
    Dim strMarshalAll As String
    Dim strMarshalModified As String

    ' Open connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Set Cnxn = New ADODB.Connection
    Cnxn.Open strCnxn
```

```
' open recordset with names from Employees table
' and set some properties through object refs
Set rstEmployees = New ADODB.Recordset
rstEmployees.CursorType = adOpenKeyset
rstEmployees.LockType = adLockOptimistic
rstEmployees.CursorLocation = adUseClient

strSQLEmployees = "SELECT fname, lname FROM Employee ORDER BY lname"

rstEmployees.Open strSQLEmployees, Cnxn, , , adCmdText
  ' empty properties have been set above

  ' Store original data
strOldFirst = rstEmployees!fname
strOldLast = rstEmployees!lname

' Change data in edit buffer
rstEmployees!fname = "Linda"
rstEmployees!lname = "Kobara"

' Show contents of buffer and get user input
strMessage = "Edit in progress:" & vbCr & _
    " Original data = " & strOldFirst & " " & _
    strOldLast & vbCr & " Data in buffer = " & _
    rstEmployees!fname & " " & rstEmployees!lname & vbCr & vbCr & _
    "Use Update to replace the original data with " & _
    "the buffered data in the Recordset?"
strMarshalAll = "Would you like to send all the rows " & _
                "in the recordset back to the server?"
strMarshalModified = "Would you like to send only " & _
                "modified rows back to the server?"

If MsgBox(strMessage, vbYesNo) = vbYes Then
    If MsgBox(strMarshalAll, vbYesNo) = vbYes Then
        rstEmployees.MarshalOptions = adMarshalAll
        rstEmployees.Update
    ElseIf MsgBox(strMarshalModified, vbYesNo) = vbYes Then
        rstEmployees.MarshalOptions = adMarshalModifiedOnly
        rstEmployees.Update
    End If
End If

' sShow the resulting data
MsgBox "Data in recordset = " & rstEmployees!fname & " " & _
    rstEmployees!lname

' restore original data because this is a demonstration
If Not (strOldFirst = rstEmployees!fname And _
        strOldLast = rstEmployees!lname) Then
    rstEmployees!fname = strOldFirst
    rstEmployees!lname = strOldLast
    rstEmployees.Update
End If

   ' clean up
rstEmployees.Close
Cnxn.Close
Set rstEmployees = Nothing
```

```
    Set Cnxn = Nothing

End Sub
'EndMarshalOptionsVB
```

4.2.2.18MaxRecords Property Example

This example uses the MaxRecords property to open a Recordset containing the 10 most expensive titles in the Titles table.

```
'BeginMaxRecordsVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub MaxRecordsX()

    Dim rstTitles As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strSQLTitles As String

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=pubs;User
Id=sa;Password=;"
    Cnxn.Open strCnxn

    ' Open recordset containing the 10 most expensive
    ' titles in the Titles table
    Set rstTitles = New ADODB.Recordset
    rstTitles.MaxRecords = 10

    strSQLTitles = "SELECT Title, Price FROM Titles ORDER BY Price DESC"
    rstTitles.Open strSQLTitles, strCnxn, adOpenStatic, adLockReadOnly, adCmdText

    ' Display the contents of the recordset
    Debug.Print "Top Ten Titles by Price:"

    Do Until rstTitles.EOF
        Debug.Print "    " & rstTitles!Title & " - " & rstTitles!Price
        rstTitles.MoveNext
    Loop

    ' clean up
    rstTitles.Close
    Cnxn.Close
    Set rstTitles = Nothing
    Set Cnxn = Nothing
End Sub
'EndMaxRecordsVB
```

4.2.2.19NumericScale and Precision Properties Example

This example uses the NumericScale and Precision properties to display the numeric scale and precision of fields in the Discounts table of the Pubs database.

```
'BeginNumericScaleVB

    'To integrate this code
```

```
    'replace the data source and initial catalog values
    'in the connection string

Public Sub NumericScaleX()

    ' connection and recordset variables
    Dim rstDiscounts As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim fldTemp As ADODB.Field
    Dim strCnxn As String
    Dim strSQLDiscounts As String

    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=pubs;User
Id=sa;Password=;"
    Cnxn.Open strCnxn

    ' Open recordset
    Set rstDiscounts = New ADODB.Recordset
    strSQLDiscounts = "Discounts"
    rstDiscounts.Open strSQLDiscounts, Cnxn, adOpenStatic, adLockReadOnly,
adCmdTable

    ' Display numeric scale and precision of
    ' numeric and small integer fields
    For Each fldTemp In rstDiscounts.Fields
        If fldTemp.Type = adNumeric Or fldTemp.Type = adSmallInt Then
            MsgBox "Field: " & fldTemp.Name & vbCr & _
                "Numeric scale: " & _
                    fldTemp.NumericScale & vbCr & _
                "Precision: " & fldTemp.Precision
        End If
    Next fldTemp

    ' clean up
    rstDiscounts.Close
    Cnxn.Close
    Set rstDiscounts = Nothing
    Set Cnxn = Nothing

End Sub
'EndNumericScaleVB
```

4.2.2.20Optimize Property Example

This example demonstrates the Field object's dynamic **Optimize** property. The **zip** field of the **Authors** table in the **Pubs** database is not indexed. Setting the Optimize property to **True** on the **zip** field authorizes ADO to build an index that improves the performance of the Find method.

```
'BeginOptimizeVB
Public Sub OptimizeX()

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

    ' recordset and connection variables
    Dim Cnxn As ADODB.Connection
```

```
    Dim rstAuthors As ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLAuthors As String

     ' Open connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Set Cnxn = New ADODB.Connection
    Cnxn.Open strCnxn

     ' open recordset client-side to enable index creation
    Set rstAuthors = New ADODB.Recordset
    rstAuthors.CursorLocation = adUseClient
    strSQLAuthors = "SELECT * FROM Authors"
    rstAuthors.Open strSQLAuthors, Cnxn, adOpenStatic, adLockReadOnly, adCmdText
     ' Create the index
    rstAuthors!zip.Properties("Optimize") = True
     ' Find Akiko Yokomoto
    rstAuthors.Find "zip = '94595'"

     ' show results
    Debug.Print rstAuthors!au_fname & " " & rstAuthors!au_lname & " " & _
                rstAuthors!address & " " & rstAuthors!city & " " & rstAuthors!State
    rstAuthors!zip.Properties("Optimize") = False   'Delete the index

     ' clean up
    rstAuthors.Close
    Cnxn.Close
    Set rstAuthors = Nothing
    Set Cnxn = Nothing
End Sub
'EndOptimizeVB
```

4.2.2.21OriginalValue and UnderlyingValue Properties Example

This example demonstrates the OriginalValue and UnderlyingValue properties by displaying a message if a record's underlying data has changed during a Recordset batch update.

```
'BeginOriginalValueVB
Public Sub OriginalValueX()

     'To integrate this code
     'replace the data source and initial catalog values
     'in the connection string

    Dim Cnxn As ADODB.Connection
    Dim rstTitles As ADODB.Recordset
    Dim fldType As ADODB.Field
    Dim strCnxn As String
    Dim strSQLTitles As String

    ' Open connection.
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

    ' Open recordset for batch update
    ' using object refs to set properties
```

```
    Set rstTitles = New ADODB.Recordset
    Set rstTitles.ActiveConnection = Cnxn
    rstTitles.CursorType = adOpenKeyset
    rstTitles.LockType = adLockBatchOptimistic
    strSQLTitles = "titles"
    rstTitles.Open strSQLTitles

    ' Set field object variable for Type field
    Set fldType = rstTitles!Type

    ' Change the type of psychology titles
    Do Until rstTitles.EOF
        If Trim(fldType) = "psychology" Then
            fldType = "self_help"
        End If
        rstTitles.MoveNext
    Loop

    ' Similate a change by another user by updating
    ' data using a command string
    Cnxn.Execute "UPDATE Titles SET type = 'sociology' " & _
        "WHERE type = 'psychology'"

    'Check for changes
    rstTitles.MoveFirst
    Do Until rstTitles.EOF
        If fldType.OriginalValue <> fldType.UnderlyingValue Then
            MsgBox "Data has changed!" & vbCr & vbCr & _
                "   Title ID: " & rstTitles!title_id & vbCr & _
                "   Current value: " & fldType & vbCr & _
                "   Original value: " & _
                fldType.OriginalValue & vbCr & _
                "   Underlying value: " & _
                fldType.UnderlyingValue & vbCr
        End If
    rstTitles.MoveNext
    Loop

    ' Cancel the update because this is a demonstration
    rstTitles.CancelBatch

    ' Restore original values
    Cnxn.Execute "UPDATE Titles SET type = 'psychology' " & _
        "WHERE type = 'sociology'"

    ' clean up
    rstTitles.Close
    Cnxn.Close
    Set rstTitles = Nothing
    Set Cnxn = Nothing

End Sub
'EndOriginalValueVB
```

4.2.2.22Prepared Property Example

This example demonstrates the Preparred property by opening two Command objects—one prepared and one not prepared.

```
'BeginPreparedVB
```

```
        'To integrate this code
        'replace the data source and initial catalog values
        'in the connection string

Public Sub PreparedX()

    Dim Cnxn As ADODB.Connection
    Dim cmd1 As ADODB.Command
    Dim cmd2 As ADODB.Command

    Dim strCnxn As String
    Dim strCmd As String
    Dim sngStart As Single
    Dim sngEnd As Single
    Dim sngNotPrepared As Single
    Dim sngPrepared As Single
    Dim intLoop As Integer

    ' Open a connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Set Cnxn = New ADODB.Connection
    Cnxn.Open strCnxn

    ' Create two command objects for the same
    ' command - one prepared and one not prepared
    strCmd = "SELECT title, type FROM Titles ORDER BY type"

    Set cmd1 = New ADODB.Command
    Set cmd1.ActiveConnection = Cnxn
    cmd1.CommandText = strCmd

    Set cmd2 = New ADODB.Command
    Set cmd2.ActiveConnection = Cnxn
    cmd2.CommandText = strCmd
    cmd2.Prepared = True

    ' Set a timer, then execute the unprepared
    ' command 20 times
    sngStart = Timer
    For intLoop = 1 To 20
        cmd1.Execute
    Next intLoop
    sngEnd = Timer
    sngNotPrepared = sngEnd - sngStart

    ' Reset the timer, then execute the prepared
    ' command 20 times
    sngStart = Timer
    For intLoop = 1 To 20
        cmd2.Execute
    Next intLoop
    sngEnd = Timer
    sngPrepared = sngEnd - sngStart

    ' Display performance results
    MsgBox "Performance Results:" & vbCr & _
        "     Not Prepared: " & Format(sngNotPrepared, _
```

```
        "##0.000") & " seconds" & vbCr & _
        "      Prepared: " & Format(sngPrepared, _
        "##0.000") & " seconds"

    ' clean up
    Cnxn.Close
    Set Cnxn = Nothing

End Sub
'EndPreparedVB
```

4.2.2.23Provider and DefaultDatabase Properties Example

This example demonstrates the Provider property by opening three Connection objects using different providers. It also uses the DefaultDatabase property to set the default database for the Microsoft ODBC Provider.

```
'BeginProviderVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection strings

Public Sub ProviderX()

    Dim Cnxn1 As ADODB.Connection
    Dim Cnxn2 As ADODB.Connection
    Dim Cnxn3 As ADODB.Connection
    Dim strCnxn As String

    ' Open a connection using the Microsoft ODBC provider
    Set Cnxn1 = New ADODB.Connection
    Cnxn1.ConnectionString = "driver={SQL Server};server=MyServer;uid=sa;pwd=;"
    Cnxn1.Open strCnxn
    Cnxn1.DefaultDatabase = "Pubs"

    ' Display the provider
    MsgBox "Cnxn1 provider: " & Cnxn1.Provider

    ' Open a connection using the Microsoft Jet provider
    Set Cnxn2 = New ADODB.Connection
    Cnxn2.Provider = "Microsoft.Jet.OLEDB.4.0"
    Cnxn2.Open "C:\Program Files\Microsoft Office\Office\Samples\northwind.mdb",
"admin", ""

    ' Display the provider.
    MsgBox "Cnxn2 provider: " & Cnxn2.Provider

    ' Open a connection using the Microsoft SQL Server provider
    Set Cnxn3 = New ADODB.Connection
    Cnxn3.Provider = "sqloledb"
    Cnxn3.Open "Data Source=MyServer;Initial Catalog=Pubs;", "sa", ""

    ' Display the provider
    MsgBox "Cnxn3 provider: " & Cnxn3.Provider

    ' clean up
    Cnxn1.Close
    Cnxn2.Close
    Cnxn3.Close
```

```
    Set Cnxn1 = Nothing
    Set Cnxn2 = Nothing
    Set Cnxn3 = Nothing

End Sub
'EndProviderVB
```

4.2.2.24Sort Property Example

This example uses the Recordset object's Sort property to reorder the rows of a Recordset derived from the Authors table of the Pubs database. A secondary utility routine prints each row.

```
'BeginSortVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub SortX()

     ' connection and recordset variables
    Dim Cnxn As New ADODB.Connection
    Dim rstAuthors As New ADODB.Recordset
    Dim strCnxn As String
    Dim strSQLAuthors As String

    Dim strTitle As String

  ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

    ' open client-side recordset to enable sort method
    Set rstAuthors = New ADODB.Recordset
    rstAuthors.CursorLocation = adUseClient
    strSQLAuthors = "SELECT * FROM Authors"
    rstAuthors.Open strSQLAuthors, Cnxn, adOpenStatic, adLockReadOnly, adCmdText

     ' sort the recordset last name ascending
    rstAuthors.Sort = "au_lname ASC, au_fname ASC"
     ' show output
    Debug.Print "Last Name Ascending:"
    Debug.Print "First Name    Last Name" & vbCr

    rstAuthors.MoveFirst
    Do Until rstAuthors.EOF
        Debug.Print rstAuthors!au_fname & " " & rstAuthors!au_lname
        rstAuthors.MoveNext
    Loop

     ' sort the recordset last name descending
    rstAuthors.Sort = "au_lname DESC, au_fname ASC"
     ' show output
    Debug.Print "Last Name Descending"
    Debug.Print "First Name    Last Name" & vbCr

    Do Until rstAuthors.EOF
```

```
        Debug.Print rstAuthors!au_fname & " " & rstAuthors!au_lname
        rstAuthors.MoveNext
    Loop

     'clean up
    rstAuthors.Close
    Cnxn.Close
    Set rstAuthors = Nothing
    Set Cnxn = Nothing

End Sub
'EndSortVB
```

This is the secondary utility routine that prints the given title, and the contents of the specified Recordset.
Attribute VB_Name = "Sort"

4.2.2.25Source Property Example

This example demonstrates the Source property by opening three Recordset objects based on different data sources.

```
'BeginSourceVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub SourceX()

    Dim Cnxn As ADODB.Connection
    Dim rstTitles As ADODB.Recordset
    Dim rstPublishers As ADODB.Recordset
    Dim rstPublishersDirect As ADODB.Recordset
    Dim rstTitlesPublishers As ADODB.Recordset
    Dim cmdSQL As ADODB.Command
    Dim strCnxn As String
    Dim strSQL As String

    ' Open a connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
    Cnxn.Open strCnxn

    ' Open a recordset based on a command object
    Set cmdSQL = New ADODB.Command
    Set cmdSQL.ActiveConnection = Cnxn
    strSQL = "Select title, type, pubdate FROM Titles ORDER BY title"
    cmdSQL.CommandText = strSQL
    Set rstTitles = cmdSQL.Execute()

    ' Open a recordset based on a table
    Set rstPublishers = New ADODB.Recordset
    strSQL = "Publishers"
    rstPublishers.Open strSQL, Cnxn, adOpenStatic, adLockReadOnly, adCmdTable
    'rstPublishers.Open strSQL, Cnxn, , , adCmdTable
     ' the above two lines of code are identical

    ' Open a recordset based on a table
```

```
    Set rstPublishersDirect = New ADODB.Recordset
    rstPublishersDirect.Open strSQL, strCnxn, , , adCmdTableDirect


    ' Open a recordset based on an SQL string.
    Set rstTitlesPublishers = New ADODB.Recordset
    strSQL = "SELECT title_ID AS TitleID, title AS Title, " & _
        "publishers.pub_id AS PubID, pub_name AS PubName " & _
        "FROM publishers INNER JOIN Titles " & _
        "ON publishers.pub_id = Titles.pub_id " & _
        "ORDER BY Title"
    rstTitlesPublishers.Open strSQL, strCnxn, , , adCmdText

    ' Use the Source property to display the source of each recordset.
    MsgBox "rstTitles source: " & vbCr & _
        rstTitles.Source & vbCr & vbCr & _
        "rstPublishers source: " & vbCr & _
        rstPublishers.Source & vbCr & vbCr & _
        "rstPublishersDirect source: " & vbCr & _
        rstPublishersDirect.Source & vbCr & vbCr & _
        "rstTitlesPublishers source: " & vbCr & _
        rstTitlesPublishers.Source

     ' clean up
    Cnxn.Close
    rstTitles.Close
    rstPublishers.Close
    rstTitlesPublishers.Close
    Set Cnxn = Nothing
    Set rstTitles = Nothing
    Set rstPublishers = Nothing
    Set rstTitlesPublishers = Nothing

End Sub
'EndSourceVB
```

4.2.2.26State Property Example

This example uses the State property to display a message while asynchronous connections are opening and asynchronous commands are executing.

```
'BeginStateVB

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

Public Sub StateX()

    Dim Cnxn1 As ADODB.Connection
    Dim Cnxn2 As ADODB.Connection
    Dim cmdChange As ADODB.Command
    Dim cmdRestore As ADODB.Command
    Dim strCnxn As String
    Dim strSQL As String

    ' Open two asynchronous connections, displaying
    ' a message while connecting
    Set Cnxn1 = New ADODB.Connection
```

```
    Set Cnxn2 = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "

    Cnxn1.Open strCnxn, , , adAsyncConnect
    Do Until Cnxn1.State <> adStateConnecting
        Debug.Print "Opening first connection...."
    Loop

    Cnxn2.Open strCnxn, , , adAsyncConnect
    Do Until Cnxn2.State <> adStateConnecting
        Debug.Print "Opening second connection...."
    Loop

    ' Create two command objects
    Set cmdChange = New ADODB.Command
    cmdChange.ActiveConnection = Cnxn1
    strSQL = "UPDATE Titles SET type = 'self_help' WHERE type = 'psychology'"
    cmdChange.CommandText = strSQL

    Set cmdRestore = New ADODB.Command
    cmdRestore.ActiveConnection = Cnxn2
    strSQL = "UPDATE Titles SET type = 'psychology' WHERE type = 'self_help'"
    cmdRestore.CommandText = strSQL

    ' Executing the commands, displaying a message
    ' while they are executing
    cmdChange.Execute , , adAsyncExecute
    Do Until cmdChange.State <> adStateExecuting
        Debug.Print "Change command executing...."
    Loop

    cmdRestore.Execute , , adAsyncExecute
    Do Until cmdRestore.State <> adStateExecuting
        Debug.Print "Restore command executing...."
    Loop

    Cnxn1.Close
    Cnxn2.Close

End Sub
'EndStateVB
```

4.2.2.27Status Property Example (Recordset)

This example uses the Status property to display which records have been modified in a batch operation before a batch update has occurred.

```
'BeginStatusRecordsetVB
Public Sub StatusX()

     'To integrate this code
     'replace the data source and initial catalog values
     'in the connection string

     ' connection and recordset variables
    Dim rstTitles As ADODB.Recordset
    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
```

```
    Dim strSQLTitles As String

     ' open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

     ' open recordset for batch update
    Set rstTitles = New ADODB.Recordset
    strSQLTitles = "Titles"
    rstTitles.Open strSQLTitles, Cnxn, adOpenKeyset, adLockBatchOptimistic,
adCmdTable

    ' change the type of psychology titles
    Do Until rstTitles.EOF
        If Trim(rstTitles!Type) = "psychology" Then rstTitles!Type = "self_help"
        rstTitles.MoveNext
    Loop

    ' display Title ID and status
    rstTitles.MoveFirst
    Do Until rstTitles.EOF
        If rstTitles.Status = adRecModified Then
            Debug.Print rstTitles!title_id & " - Modified"
        Else
            Debug.Print rstTitles!title_id
        End If
    rstTitles.MoveNext
    Loop

     ' Cancel the update because this is a demonstration
    rstTitles.CancelBatch

     ' clean up
    rstTitles.Close
    Cnxn.Close
    Set rstTitles = Nothing
    Set Cnxn = Nothing

End Sub
'EndStatusRecordsetVB
```

4.2.2.28StayInSync Property Example

This example demonstrates how the StayInSync property facilitates accessing rows in a hierarchical Recordset.

The outer loop displays each author's first and last name, state, and identification. The appended Recordset for each row is retrieved from the Fields collection and automatically assigned to rstTitleAuthor by the StayInSync property whenever the parent Recordset moves to a new row. The inner loop displays four fields from each row in the appended recordset.

```
'BeginStayInSyncVB
Public Sub StayInSyncX()

    'To integrate this code create a DSN called Pubs
    'with a user_id = sa and no password

    Dim Cnxn As ADODB.Connection
```

```
    Dim rst As ADODB.Recordset
    Dim rstTitleAuthor As New ADODB.Recordset
    Dim strCnxn As String

        ' open connection with Data Shape attributes
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=;"
    Cnxn.Open strCnxn

        ' create recordset
    Set rst = New ADODB.Recordset
    rst.StayInSync = True
    rst.Open "SHAPE {select * from Authors} " & _
                    "APPEND ({select * from titleauthor}" & _
                    "RELATE au_id TO au_id) AS chapTitleAuthor", _
                    Cnxn, , , adCmdText

    Set rstTitleAuthor = rst("chapTitleAuthor").Value
    Do Until rst.EOF
        Debug.Print rst!au_fname & " " & rst!au_lname & " " & _
                    rst!State & " " & rst!au_id

        Do Until rstTitleAuthor.EOF
            Debug.Print rstTitleAuthor(0) & " " & rstTitleAuthor(1) & " " & _
                    rstTitleAuthor(2) & " " & rstTitleAuthor(3)
        rstTitleAuthor.MoveNext
        Loop

    rst.MoveNext
    Loop

     ' clean up
    rst.Close
    Cnxn.Close
    Set rst = Nothing
    Set Cnxn = Nothing


End Sub
'EndStayInSyncVB
```

4.2.2.29Type Property Example (Filed)

This example demonstrates the Type property by displaying the name of the constant that corresponds to the value of the Type property of all the Field objects in the Employees table. The FieldType function is required for this procedure to run.

```
'BeginTypeFieldVB
Public Sub TypeX()

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

   Dim Cnxn As ADODB.Connection
   Dim rstEmployees As ADODB.Recordset
   Dim fld As ADODB.Field
   Dim strCnxn As String
   Dim strSQLEmployee As String
   Dim FieldType As String
```

```
    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
    Cnxn.Open strCnxn

    ' Open recordset with data from Employees table
    Set rstEmployees = New ADODB.Recordset
    strSQLEmployee = "employee"
    rstEmployees.Open strSQLEmployee, Cnxn, , , adCmdTable
    'rstEmployees.Open strSQLEmployee, Cnxn, adOpenStatic, adLockReadOnly, adCmdTable
     ' the above two lines of code are identical

    Debug.Print "Fields in Employees Table:" & vbCr

    ' Enumerate Fields collection of Employees table
    For Each fld In rstEmployees.fields

        ' translate field-type code to text
      Select Case fld.Type
        Case adChar
            FieldType = "adChar"
        Case adVarChar
            FieldType = "adVarChar"
        Case adSmallInt
            FieldType = "adSmallInt"
        Case adUnsignedTinyInt
            FieldType = "adUnsignedTinyInt"
        Case adDBTimeStamp
            FieldType = "adDBTimeStamp"
      End Select
        ' show results
      Debug.Print "    Name: " & fld.Name & vbCr & _
           "    Type: " & FieldType & vbCr

    Next fld

End Sub
'EndTypeFieldVB

Attribute VB_Name = "TypeField"
```

4.2.2.30Type Property Example (Property)

This example demonstrates the Type property. It is a model of a utility for listing the names and types of a collection, like Properties, Fields, etc.
We do not need to open the Recordset to access its **Properties** collection; they come into existence when the **Recordset** object is instantiated. However, setting the CursorLocation property to **adUseClient** adds several dynamic properties to the **Recordset** object's **Properties** collection, making the example a little more interesting. For sake of illustration, we explicitly use the Item property to access each Property object.

```
'BeginTypePropertyVB
Public Sub TypeX()

     ' recordset variables
    Dim rst As ADODB.Recordset
    Dim prop As ADODB.Property
     ' property variables
```

```
Dim ix As Integer
Dim strMsg As String

 ' create client-side recordset
Set rst = New ADODB.Recordset
rst.CursorLocation = adUseClient
 ' enumerate property types
For ix = 0 To rst.Properties.count - 1
      Set prop = rst.Properties.Item(ix)
      Select Case prop.Type
          Case adBigInt
              strMsg = "adBigInt"
          Case adBinary
              strMsg = "adBinary"
          Case adBoolean
              strMsg = "adBoolean"
          Case adBSTR
              strMsg = "adBSTR"
          Case adChapter
              strMsg = "adChapter"
          Case adChar
              strMsg = "adChar"
          Case adCurrency
              strMsg = "adCurrency"
          Case adDate
              strMsg = "adDate"
          Case adDBDate
              strMsg = "adDBDate"
          Case adDBTime
              strMsg = "adDBTime"
          Case adDBTimeStamp
              strMsg = "adDBTimeStamp"
          Case adDecimal
              strMsg = "adDecimal"
          Case adDouble
              strMsg = "adDouble"
          Case adEmpty
              strMsg = "adEmpty"
          Case adError
              strMsg = "adError"
          Case adFileTime
              strMsg = "adFileTime"
          Case adGUID
              strMsg = "adGUID"
          Case adIDispatch
              strMsg = "adIDispatch"
          Case adInteger
              strMsg = "adInteger"
          Case adIUnknown
              strMsg = "adIUnknown"
          Case adLongVarBinary
              strMsg = "adLongVarBinary"
          Case adLongVarChar
              strMsg = "adLongVarChar"
          Case adLongVarWChar
              strMsg = "adLongVarWChar"
          Case adNumeric
              strMsg = "adNumeric"
          Case adPropVariant
```

```
                strMsg = "adPropVariant"
            Case adSingle
                strMsg = "adSingle"
            Case adSmallInt
                strMsg = "adSmallInt"
            Case adTinyInt
                strMsg = "adTinyInt"
            Case adUnsignedBigInt
                strMsg = "adUnsignedBigInt"
            Case adUnsignedInt
                strMsg = "adUnsignedInt"
            Case adUnsignedSmallInt
                strMsg = "adUnsignedSmallInt"
            Case adUnsignedTinyInt
                strMsg = "adUnsignedTinyInt"
            Case adUserDefined
                strMsg = "adUserDefined"
            Case adVarBinary
                strMsg = "adVarBinary"
            Case adVarChar
                strMsg = "adVarChar"
            Case adVariant
                strMsg = "adVariant"
            Case adVarNumeric
                strMsg = "adVarNumeric"
            Case adVarWChar
                strMsg = "adVarWChar"
            Case adWChar
                strMsg = "adWChar"
            Case Else
                strMsg = "*UNKNOWN*"
        End Select
            'show results
        Debug.Print "Property " & ix & ": " & prop.Name & _
                    ", Type = " & strMsg
    Next ix

End Sub
'EndTypePropertyVB
```

4.2.2.31Value Property Example

This example demonstrates the Value property with Field and Property objects by displaying field and property values for the Employees table.

```
'BeginValueVB
Public Sub ValueX()

    'To integrate this code
    'replace the data source and initial catalog values
    'in the connection string

    ' connection and recordset variables
  Dim rstEmployees As ADODB.Recordset
  Dim Cnxn As ADODB.Connection
  Dim strCnxn As String
  Dim strSQLEmployees As String
    ' field property variables
  Dim fld As ADODB.Field
  Dim prp As ADODB.Property
```

```
    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User Id=sa;Password=; "
    Cnxn.Open strCnxn

   ' Open recordset with data from Employees table
    Set rstEmployees = New ADODB.Recordset
    strSQLEmployees = "employee"
    rstEmployees.Open strSQLEmployees, Cnxn, , , adCmdTable
    'rstEmployees.Open strSQLEmployees, Cnxn, adOpenStatic, adLockReadOnly, adCmdTable
     ' the above two lines of code are identical

    Debug.Print "Field values in rstEmployees"

    ' Enumerate the Fields collection of the Employees table
    For Each fld In rstEmployees.fields
        ' Because Value is the default property of a
        ' Field object, the use of the actual keyword
        ' here is optional.
        Debug.Print "     " & fld.Name & " = " & fld.Value
    Next fld

    Debug.Print "Property values in rstEmployees"

    ' Enumerate the Properties collection of the Recordset object
    For Each prp In rstEmployees.Properties
        Debug.Print "     " & prp.Name & " = " & prp.Value
        ' because Value is the default property of a Property object
        ' use of the actual Value keyword is optional
    Next prp

     ' clean up
    rstEmployees.Close
    Cnxn.Close
    Set rstEmployees = Nothing
    Set Cnxn = Nothing

End Sub
'EndValueVB
```

4.2.2.32Version Property Example

This example uses the Version property of a Connection object to display the current ADO version. It also uses several dynamic properties to show:
The name and version of current DBMS.
OLE DB version.
Provider name and version.
ODBC version.
ODBC driver name and version.

```
'BeginVersionVB
Public Sub VersionX()

    Dim Cnxn As ADODB.Connection
    Dim strCnxn As String
    Dim strVersionInfo As String
```

```
    ' Open connection
    Set Cnxn = New ADODB.Connection
    strCnxn = "Provider=sqloledb;Data Source=MyServer;Initial Catalog=Pubs;User
Id=sa;Password=; "
    Cnxn.Open strCnxn

    strVersionInfo = "ADO Version: " & Cnxn.Version & vbCr
    strVersionInfo = strVersionInfo & "DBMS Name: " & Cnxn.Properties("DBMS Name") &
vbCr
    strVersionInfo = strVersionInfo & "DBMS Version: " & Cnxn.Properties("DBMS Version")
& vbCr
    strVersionInfo = strVersionInfo & "OLE DB Version: " & Cnxn.Properties("OLE DB
Version") & vbCr
    strVersionInfo = strVersionInfo & "Provider Name: " & Cnxn.Properties("Provider
Name") & vbCr
    strVersionInfo = strVersionInfo & "Provider Version: " & Cnxn.Properties("Provider
Version") & vbCr

    MsgBox strVersionInfo

    Cnxn.Close
    Set Cnxn = Nothing

End Sub
'EndVersionVB
```

Index