Data Access Objects Overview

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dahowDataAccessOverviewC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dahowDataAccessOverviewX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dahowDataAccessOverviewS"}

Data Access Objects (DAO) enables you to use a programming language to access and manipulate data in local or remote databases, and to manage databases, their objects, and their structure.

Object Models

DAO supports two different database environments, or "workspaces."

- Microsoft Jet workspaces allow you to access data in Microsoft Jet databases, Microsoft Jet-connected ODBC databases, and installable ISAM data sources in other formats, such as Paradox or Lotus 1-2-3.
- ODBCDirect workspaces allow you to access database servers through ODBC, without loading the Microsoft Jet database engine.

Use the Microsoft Jet workspace when you open a Microsoft Jet database (.mdb file) or other desktop ISAM database, or when you need to take advantage of Microsoft Jet's unique features, such as the ability to join data from different database formats.

The ODBCDirect workspace provides an alternative when you only need to execute queries or stored procedures against a back-end server, such as Microsoft SQL Server, or when your client application needs the specific capabilities of ODBC, such as batch updates or asynchronous query execution.

DAO Objects

There are 17 different DAO object types. You can declare new DAO object variables for any of the object types.

For example, the following Visual Basic for Applications (VBA) code creates object variables for a **Database** object, a dynaset-type **Recordset** object, and a **Field** object:

```
Dim dbsExample As Database
Dim rstExample As Recordset
Dim fldExample As Field

Set dbsExample = OpenDatabase("Biblio.mdb")
Set rstExample = dbsExample.OpenRecordset("Authors", _     dbOpenDynaset)
Set fldExample = rstExample.Fields("Au_ID")
```

DAO Collections

Each DAO object type other than **DBEngine** also has a corresponding collection. A collection includes all the existing objects of that type. For example, the **Recordsets** collection contains all open **Recordset** objects. Each collection is "owned" by another object at the next higher level in the hierarchy. A **Database** object "owns" a **Recordsets** collection. Except for the **Connection** and **Error** objects, every DAO object has a **Properties** collection.

Most DAO objects have default collections and default properties. For example, the default collection of a **Recordset** object is the **Fields** collection and the default property of a **Field** object is the **Value** property. You can simplify your code by taking advantage of these defaults. For example, the following code sets the value of the PubID field in the current record:

```
rstExample!PubID=99
```

DBEngine and Workspace Objects

All DAO objects are derived from the **DBEngine** object. You can set the **DefaultType** property on the **DBEngine** object to determine the workspace type (Microsoft Jet or ODBCDirect) to create on

subsequent **CreateWorkspace** method calls, or you can override this property with the *type* argument in the **CreateWorkspace** method itself. When your application creates a workspace, the appropriate library — the Microsoft Jet database engine or ODBC — is loaded into memory at that time.

You can open additional **Workspace** objects as needed. Each **Workspace** object has a user ID and password associated with it.

Using the Microsoft Jet Workspace

Opening a Database

To open a database, you simply open an existing **Database** object, or create a new one. This object can represent a Microsoft Jet database (.mdb file), an ISAM database (for example, Paradox), or an ODBC database connected through the Microsoft Jet database engine (also known as a "Microsoft Jet-connected ODBC database").

Data-Definition Language

You can use object variables and other DDL features to modify your database structure. For example, you can add a new **Field** object to an existing table with the following code:

```
Dim dbs As Database, tdf As TableDef, fld As Field
' Open a database.
Set dbs = OpenDatabase("Biblio.mdb")
' Open a TableDef.
Set tdf = dbs.TableDefs("Authors")
' Create a new field.
Set fld = tdf.CreateField("Address", dbText, 20)
' Append field to the TableDef Fields collection.
tdf.Fields.Append fld
```

This code creates a new object variable for a **Field** object and adds it to a **TableDef** object with the **Append** method. Because a **TableDef** object contains the definition of a table, the table now has a field named Address for entering data. In much the same way, you can create new tables and new indexes.

Data Manipulation

DAO provides an excellent set of data manipulation tools. You can create a **Recordset** object to conveniently query a database and manipulate the resulting set of records. The **OpenRecordset** method accepts an SQL string, or a **QueryDef** (stored query) name as a data source argument, or it can be opened from a **QueryDef** object or a **TableDef** object, using that object as its data source. The resulting **Recordset** object features an extremely rich set of properties and methods with which to browse and modify data.

The **Recordset** object is available in four different types — Table, Dynaset, Forward-Only, and Snapshot.

Transactions

All **Database** objects opened against a **Workspace** object share a common transaction scope. That is, when you use the **BeginTrans** method on a **Workspace** object, it applies to all open databases within that **Workspace** object. In the same way, when you use the **CommitTrans** method against the **Workspace**, it applies to all open databases in the **Workspace** object.

Replication

You can use database replication to create and maintain replicas of a master Microsoft Jet database, using the **Synchronize** method to periodically update all or part of the replicas, or to copy new data from one replica to another. You can also restrict the update to only selected records, using the

ReplicaFilter property, and then synchronize those records with the **PopulatePartial** method.

Security

You can restrict access to one or more .mdb databases or their tables using security settings established and managed by the Microsoft Jet database engine. In your code, you can establish **Group** and **User** objects to define the scope and level of permissions available to individual users on an object-by-object basis. For example, you can establish permissions for a specific user to provide read-only access to one table and full access to another.

Using the ODBCDirect Object Model

Connecting to a Database

A **Connection** object is similar to a **Database** object. In fact, a **Connection** object and a **Database** object represent different references to the same object, and properties on each of these two object types allow you to obtain a reference to the other corresponding object, which simplifies the task of converting ODBC client applications that use Microsoft Jet to use ODBCDirect instead. Use the **OpenConnection** method to connect to an ODBC data source. The resulting **Connection** object contains information about the connection, such as the server name, the data source name, and so on.

Queries

Although DAO does not support stored queres in an ODBCDirect workspace, a compiled query can be created as a **QueryDef** object and used to execute action queries, and can also be used to execute stored procedures on the server. The **Prepare** property lets you decide whether to create a private, temporary stored procedure on the server from a **QueryDef** before actually executing the query.

Parameter queries can also be passed to the server, using **Parameter** objects on the **QueryDef**. The **Direction** property lets you specify a **Parameter** as input, output, or both, or to accept a return value from a stored procedure.

Data Manipulation

Creating a **Recordset** object is a convenient way to query a database and manipulate the resulting set of records. The **OpenRecordset** method accepts an SQL string, or a **QueryDef** object (stored query) as a data source argument. The resulting **Recordset** object features an extremely rich set of properties and methods with which to browse and modify data.

The **Recordset** object is available in four different types — Dynamic, Dynaset, Forward-Only, and Snapshot — corresponding to ODBC cursor types — Dynamic, Keyset, Forward-only, and Static.

A batch update cursor library is available for client applications that need to work with a cursor without holding locks on the server or without issuing update requests one record at a time. Instead, the client stores update information on many records in a local buffer (or "batch"), and then issues a batch update.

Asynchronous Method Execution

The **Execute**, **MoveLast**, **OpenConnection**, and **OpenRecordset** methods feature the **dbRunAsync** option. This allows your client application to do other tasks (such as loading forms, for example) while the method is executing. You can check the **StillExecuting** property to see whether the task is complete, and terminate an asynchronous task with the **Cancel** method.

Data Access Methods by Object

{ewc HLP95EN.dll, DYNALINK, "See Also":"daidxMethodsC "} {ewc HLP95EN.dll, DYNALINK, "Specifics":"daidxMethodsS "}

This reference groups all DAO methods by object. To see whether a particular method is available for Microsoft Jet or ODBC workspaces, check the Help topic for that method.

Connection
Container — no methods
Database
DBEngine
Document
Error — no methods
Field
Group
Index
Parameter — no methods
Property — no methods
QueryDef
Recordset
Relation
TableDef
User
Workspace

Data Access Methods for Microsoft Jet Workspaces

{ewc HLP95EN.dll, DYNALINK, "See Also":"daidxMethodsReferenceJetC "} {ewc HLP95EN.dll, DYNALINK, "Specifics":"daidxMethodsReferenceJetS "}

This reference lists alphabetically all DAO methods available for Microsoft Jet workspaces (ISAM database files).

A-C

AddNew
Append
AppendChunk
BeginTrans
CancelUpdate
Clone
Close
CommitTrans
CompactDatabase
CopyQueryDef
CreateDatabase
CreateField
CreateGroup
CreateIndex
CreateProperty
CreateQueryDef
CreateRelation
CreateTableDef
CreateUser
CreateWorkspace

D-M

Delete
Edit
Execute
FillCache
FindFirst
FindLast
FindNext
FindPrevious
GetChunk
GetRows
Idle
MakeReplica
Move
MoveFirst
MoveLast
MoveNext
MovePrevious

N-Z

NewPassword
OpenDatabase
OpenRecordset
PopulatePartial
Refresh
RefreshLink
RegisterDatabase
RepairDatabase
Requery
Rollback
Seek
SetOption
Synchronize
Update

Data Access Methods for ODBCDirect Workspaces

{ewc HLP95EN.dll, DYNALINK, "See Also":"daidxMethodsReferenceODBCC "} {ewc HLP95EN.dll, DYNALINK, "Specifics":"daidxMethodsReferenceODBCS "}

This reference alphabetically lists all DAO methods available for ODBCDirect workspaces.

A-C

AddNew
Append
AppendChunk
BeginTrans
Cancel
CancelUpdate
Clone
Close
CommitTrans
CreateQueryDef
CreateWorkspace

D-M

Delete
Edit
Execute
GetChunk
GetRows
Move
MoveFirst
MoveLast
MoveNext
MovePrevious

N-Z

NextRecordset
OpenConnection
OpenDatabase
OpenRecordset
Refresh
RegisterDatabase
Requery
Rollback
Update

Data Access Object Model for Microsoft Jet Workspaces

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daconmsjetdatabaseengine25c"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daconMSJetDatabaseEngine25S"}

DBEngine

Errors

Error

Workspaces

Workspace

Databases

Database

Containers

Container

Documents

Document

QueryDefs

QueryDef

Fields

Field

Parameters

Parameter

Recordsets

Recordset

Fields

Field

Relations

Relation

Fields

Field

TableDefs

TableDef

Fields

Field

Indexes

Index

Fields

Field

Groups

Group

Users

User

Users

User

Groups

Group

Data Access Object Model for ODBCDirect Workspaces

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daconMSJetDatabaseEngine35C"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daconMSJetDatabaseEngine35S"}

Error

Workspace

Connections

Connection

QueryDefs

Parameter

Recordsets

Field

Data Access Objects and Collections Reference

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daidxObjectsandCollectionsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daidxObjectsandCollectionsS"}

DAO objects and collections provide a framework for using code to create and manipulate components of your database system. Objects and collections have properties that describe the characteristics of database components and methods that you use to manipulate them. Together these objects and collections form a hierarchical model of your database structure, which you can control programmatically.

Objects and collections provide different types of containment relations: Objects contain zero or more collections, all of different types; and collections contain zero or more objects, all of the same type. Although objects and collections are similar entities, the distinction differentiates the two types of relations.

In the following table, the type of collection in the first column contains the type of object in the second column. The third column describes what each type of object represents.

Collection	Object	Description
Connections	Connection	Information about a connection to an ODBC data source (ODBCDirect workspaces only)
Containers	Container	Storage for information about a predefined object type (Microsoft Jet workspaces only)
Databases	Database	An open database
None	DBEngine	The Microsoft Jet database engine
Documents	Document	Information about a saved, predefined object (Microsoft Jet workspaces only)
Errors	Error	Information about any errors associated with this object
Fields	Field	A column that is part of a table, query, index, relation, or recordset
Groups	Group	A group of user accounts (Microsoft Jet workspaces only)
Indexes	Index	Predefined ordering and uniqueness of values in a table (Microsoft Jet workspaces only)
Parameters	Parameter	A parameter for a parameter query
Properties	Property	A built-in or user-defined property
QueryDefs	QueryDef	A saved query definition
Recordsets	Recordset	The records in a base

		table or query
Relations	Relation	A relationship between fields in tables and queries (Microsoft Jet workspaces only)
TableDefs	TableDef	A saved table definition (Microsoft Jet workspaces only)
Users	User	A user account (Microsoft Jet workspaces only)
Workspaces	Workspace	A session of the Microsoft Jet database engine

Data Access Properties by Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daidxPropertiesC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daidxPropertiesS"}

This reference groups all DAO properties by object or collection. To determine whether a particular property is available to Microsoft Jet or ODBC databases, check the Help topic for that property.

Connection
Container
Database
DBEngine
Document
Error
Field
Group
Index
Parameter
Property
QueryDef
Recordset
Relation
TableDef
User
Workspace

Data Access Properties for Microsoft Jet Workspaces

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daidxPropertiesReferenceJetC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daidxPropertiesReferenceJetS"}

This reference lists alphabetically all DAO properties available to Microsoft Jet workspaces.

A-C

AbsolutePosition
AllowZeroLength
AllPermissions
Attributes
BOF
Bookmark
Bookmarkable
CacheSize
CacheStart
Clustered
CollatingOrder
ConflictTable
Connect
Container
Count

D-H

DataUpdatable
DateCreated
DefaultUser
DefaultPassword
DefaultValue
Description
DesignMasterID
DistinctCount
EditMode
EOF
FieldSize
Filter
Foreign
ForeignName
ForeignTable
HelpContext
HelpFile

I-O

IgnoreNulls
Index
Inherit
Inherited
IniPath
IsolateODBCTrans
KeepLocal
LastModified
LastUpdated
LockEdits
LoginTimeout
LogMessages
MaxRecords
Name
NoMatch
Number
ODBCTimeout
OrdinalPosition
Owner

P-R

PartialReplica
Password
PercentPosition
Permissions
PID
RecordsAffected
Replicable
ReplicableBool
ReplicaFilter
ReplicaID

Primary
QueryTimeout
RecordCount
Required
Restartable
ReturnsRecords

S-Z

Size
Sort
Source
SourceField
SourceTable
SourceTableName
SQL
SystemDB
Table
Transactions
Type
Unique
Updatable
UserName
V1xNullBehavior
ValidateOnSet
ValidationRule
ValidationText
Value
Version

Data Access Properties for ODBCDirect Workspaces

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daidxPropertiesReferenceODBCC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daidxPropertiesReferenceODBCS"}

This reference lists alphabetically all DAO properties available to ODBCDirect workspaces.

A-D

AbsolutePosition
Attributes
BatchCollisionCount
BatchCollisions
BatchSize
BOF
Bookmark
Bookmarkable
CacheSize
Connect
Connection
Count
Database
DataUpdatable
DefaultCursorDriver
DefaultType
Description
Direction

E-Q

EditMode
EOF
FieldSize
HelpContext
HelpFile
LastModified
LockEdits
LoginTimeout
LogMessages
MaxRecords
Name
Number
ODBCTimeout
OrdinalPosition
OriginalValue
PercentPosition
Prepare
QueryTimeout

R-Z

RecordCount
RecordsAffected
RecordStatus
Restartable
Size
Source
SourceField
SourceTable
SQL
StillExecuting
Transactions
Type
Updatable
UpdateOptions
UserName
Value
Version
VisibleValue

What's New in DAO?

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daidxWhatsNewInDAOC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daidxWhatsNewInDAOS"}

DAO 3.5 introduces a new client/server connection mode, called "ODBCDirect." ODBCDirect establishes a connection directly to an ODBC data source, without loading the Microsoft Jet database engine into memory, and is useful in situations where specific features of ODBC are required.

For Microsoft Jet databases, there are also new interfaces to expose Microsoft Jet's new partial replication feature.

Note You can send DAO queries to a variety of different database servers with ODBCDirect, and different servers will recognize slightly different dialects of SQL. Therefore, context-sensitive Help is no longer provided for Microsoft Jet SQL, although online Help for Microsoft Jet SQL is still included through the Help menu. Be sure to check the appropriate reference documentation for the SQL dialect of your database server when using either ODBCDirect connections or pass-through queries in Microsoft Jet-connected client/server applications.

New DAO 3.5 Interfaces for ODBCDirect

- **Connection** object — A connection to an ODBC database.
- **Cancel** method (on **Connection**, **QueryDef**, and **Recordset** objects) — Cancels execution of an asynchronous operation.
- **NextRecordset** method (on **Recordset** objects) — Retrieves the next set of records, if any, returned by a query that returned multiple sets of records in an **OpenRecordset** call, and indicates whether it successfully retrieved another set of records.
- **OpenConnection** method (on **Workspace** objects) — Opens a **Connection** object on an ODBC data source.
- **BatchCollisionCount** property (on **Recordset** objects) — Returns the number of records that did not complete during the last batch update.
- **BatchCollisions** property (on **Recordset** objects) — Returns an array of bookmarks indicating the rows that generated collisions in the last batch update.
- **BatchSize** property (on **Recordset** objects) — Sets or returns the number of statements sent back to the server in each batch.
- **Connection** property (on **Database** and **Recordset** objects) — Returns the **Connection** object that corresponds to the **Database**, or that owns the **Recordset**.
- **Database** property (on **Connection** objects) — Returns the name of the **Database** object that corresponds to the **Connection**.
- **DefaultCursorDriver** property (on **Workspace** objects) — Sets or returns the type of cursor driver used for ODBCDirect **Recordset** objects.
- **DefaultType** property (on **DBEngine** object) — Indicates what type of workspace (Microsoft Jet or ODBCDirect) will be created by the next **CreateWorkspace** method call.
- **Direction** property (on **Parameter** objects) — Indicates whether a **Parameter** object represents an input parameter, an output parameter, or both, or if the parameter is the return value from a stored procedure.
- **MaxRecords** property (on **QueryDef** objects) — Sets or returns the maximum number of records to return from a query.
- **OriginalValue** property (on **Field** objects) — Returns the value of a **Field** in the database that existed when the last batch update began.
- **Prepare** property (on **QueryDef** objects) — Returns a value that indicates whether the query should be prepared on the server as a temporary stored procedure with the ODBC **SQLPrepare** function prior to execution, or just executed using the ODBC **SQLExecDirect** function.
- **RecordStatus** property (on **Recordset** objects) — Returns a value that indicates the update status

of the current record if it is part of a batch update.

- **StillExecuting** property (on **Connection**, **QueryDef**, and **Recordset** objects) — Returns a value indicating whether or not an asynchronous operation has finished executing.
- **UpdateOptions** property (on **Recordset** objects) — Returns a value that indicates how the WHERE clause is constructed for each record during a batch update, and how the update should be executed.
- **VisibleValue** property (on **Recordset** objects) — Returns a value currently in the database that is newer than the **OriginalValue** property as determined by a batch update conflict.

New Capabilities with ODBCDirect

Server Connections

Available only in the ODBCDirect object model, the new **Connection** object contains information about a connection to an ODBC data source, such as the server name, the data source name, and so on. It is similar to a **Database** object, and will look very familiar if you've ever opened a **Database** object on an ODBC data source. In fact, a **Connection** object and a **Database** object represent different references to the same object, and new properties on each of these two object types allow you to obtain a reference to the other corresponding object, which simplifies the task of converting existing ODBC client applications that use Microsoft Jet to use ODBCDirect instead.

Batch Updates

A new batch update cursor is available for client applications that need to work with a cursor without holding locks on the server or issue update requests one record at a time. Instead, the client stores update information on many records in a local buffer (or "batch"), and then issues a batch update.

Because of the time lag between opening a **Recordset** and sending a batch of updates from that **Recordset** back to the server, other users have an opportunity to change the original data before your changes are sent to the server, so your changes "collide" with another user's changes. Several new features are available to help you determine where such collisions have occurred, following a batch update, and give you some options for resolving them.

Asynchronous Method Execution

The **Execute**, **MoveLast**, **OpenConnection**, and **OpenRecordset** methods feature the **dbRunAsync** option. This allows the client application to do other tasks (such as loading forms, and so on) while the method is executing. You can also poll to see whether the task is complete, and terminate an asynchronous task.

Client Support for ODBC Cursors

Four different **Recordset** types support the following ODBC cursor types:

ODBC Cursor	Recordset type
Dynamic	**dbOpenDynamic** (New in DAO 3.5)
Dynaset	**dbOpenDynaset**
Forward-Only	**dbOpenForwardOnly** (New in DAO 3.5)
Static	**dbOpenSnapshot**

New DAO 3.5 Interfaces for the Microsoft Jet Database Engine

- **PopulatePartial** method (on **Database** objects) — Synchronizes any changes in a partial replica with the full replica, clears all records in the partial replica, and then repopulates the partial replica based on the current replica filters.
- **SetOption** method (on **DBEngine** object) — Overrides the registry values for the Microsoft Jet database engine for the duration of the current instance of DAO.

- **FieldSize** property (on **Field** objects) — Replaces the **FieldSize** method. Syntactically, their usage is the same, so this will not require changes to your existing code.
- **MaxRecords** property (on **QueryDef** objects) — Sets or returns the maximum number of records to return from a query.
- **ReplicaFilter** property (on **TableDef** objects) — Returns a value that indicates which subset of records is replicated to that table from a full replica.
- **PartialReplica** property (on **Relation** objects) — Indicates which **Relation** object should be considered when populating a partial replica from a full replica.

New Capabilities with the Microsoft Jet Database Engine

Partial Replication

Version 3.5 of the Microsoft Jet database engine allows users to replicate portions of a table instead of the whole table (only row restrictions are permitted, not columns). There are two types of filters used in a partial replica — Boolean and relationship. Boolean filters select only rows that meet a certain criteria to limit the rows in a table that are replicated. DAO represents this filter with the **ReplicaFilter** property on a **TableDef**. Relationship filters enforce a relationship between partially replicated tables to limit the rows in a table that are replicated. With DAO, you can set the **PartialReplica** property on a **Relation** which allows that **Relation** to be used in partial replication.

New Recordset Type

In DAO 3.5, **dbOpenForwardOnly** is a new *type* argument for the **OpenRecordset** method. This new **Recordset** type behaves in the same way as a DAO 3.0 snapshot-type **Recordset** opened with the **dbForwardOnly** option.

Run-time Registry Override

The new **SetOption** method allows you to override Microsoft Jet Registry settings at run time. This lets you fine tune Microsoft Jet query performance, timeout delays, and so on.

Obsolete Features in DAO

{ewc HLP95EN.dll, DYNALINK, "See Also":"dahowObsoleteFeaturesC"} {ewc HLP95EN.dll, DYNALINK, "Example":"dahowObsoleteFeaturesX":1} {ewc HLP95EN.dll, DYNALINK, "Specifics":"dahowObsoleteFeaturesS"}

Microsoft Access versions 1.x and 2.0 and Microsoft Visual Basic version 3.0 used earlier versions of DAO. Several objects, methods, properties, and statements in those earlier versions are considered "obsolete" but are still supported for backwards compatibility with existing user code.

The following is a list of DAO methods, properties, objects, and statements that have been replaced by more powerful, flexible, and easy-to-use features. Each obsolete feature in the list has a corresponding replacement feature.

Obsolete feature	Replacement feature
All **CreateDynaset** methods	**OpenRecordset** method
All **CreateSnapshot** methods	**OpenRecordset** method
All **ListFields** methods	**Fields** collection
All **ListIndexes** methods	**Indexes** collection
CompactDatabase statement	**DBEngine.CompactDatabase** method
CreateDatabase statement	**DBEngine.CreateDatabase** method
DBEngine.FreeLocks method	**DBEngine.Idle** method
DBEngine.SetDefaultWorkspace method	**DBEngine.DefaultUser** and **DBEngine.Password** properties
DBEngine.SetDataAccessOption method	**DBEngine.IniPath** property
Database.BeginTrans method	**Workspace.BeginTrans** method
Database.CommitTrans method	**Workspace.CommitTrans** method
Database.Rollback method	**Workspace.Rollback** method
Database.DeleteQuerydef method	**Delete** method
Database.ExecuteSQL method	**Execute** method
Database.ListTables method	**Tabledefs** collection
Database.OpenQuerydef method	**Querydefs** collection
Database.OpenTable method	**OpenRecordset** method
FieldSize method	**FieldSize** property
Index.Fields property	**Index.Fields** collection
OpenDatabase statement	**DBEngine.OpenDatabase** method
Querydef.ListParameters method	**Parameters** collection
Snapshot object	**Recordset** object
Dynaset object	**Recordset** object
Table object	**Recordset** object

Container Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjContainerC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjContainerX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjContainerP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjContainerM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjContainerS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjContainerU":1}

A **Container** object groups similar types of **Document** objects together.

Remarks

Each **Database** object has a **Containers** collection consisting of built-in **Container** objects. Applications can define their own document types and corresponding containers (Microsoft Jet databases only); however, these objects may not always be supported through DAO.

Some of these **Container** objects are defined by the Microsoft Jet database engine while others may be defined by other applications. The following table lists the name of each **Container** object defined by the Microsoft Jet database engine and what type of information it contains.

Container name	Contains information about
Databases	Saved databases
Tables	Saved tables and queries
Relations	Saved relationships

Note Don't confuse the **Container** objects listed in the preceding table with the collections of the same name. The Databases **Container** object refers to all saved database objects, but the **Databases** collection refers only to database objects that are open in a particular workspace.

Each **Container** object has a **Documents** collection containing **Document** objects that describe instances of built-in objects of the type specified by the **Container**. You typically use a **Container** object as an intermediate link to the information in the **Document** object. You can also use the **Containers** collection to set security for all **Document** objects of a given type.

With an existing **Container** object, you can:

- Use the **Name** property to return the predefined name of the **Container** object.

- Use the **Owner** property to set or return the owner of the **Container** object. To set the **Owner** property, you must have write permission for the **Container** object, and you must set the property to the name of an existing **User** or **Group** object.
- Use the **Permissions** and **UserName** properties to set access permissions for the **Container** object; any **Document** object created in the **Documents** collection of a **Container** object inherits these access permission settings.

Because **Container** objects are built-in, you can't create new **Container** objects or delete existing ones.

To refer to a **Container** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

Containers(0)
Containers("*name*")
Containers![*name*]

Containers Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolContainerC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolContainerX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolContainerP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolContainerM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolContainerS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolContainerU":1}

A **Containers** collection contains all of the **Container** objects that are defined in a database (Microsoft Jet databases only).

Remarks

Each **Database** object has a **Containers** collection consisting of built-in **Container** objects. Some of these **Container** objects are defined by the Microsoft Jet database engine while others may be defined by other applications.

Container Object, Containers Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumContainerC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumContainerS"}

Container Object

A **Container** object contains no methods; it contains these collections and properties.

Collections

Documents (Default)

Properties

Properties

AllPermissions

Inherit

Name

Owner

Permissions

UserName

Containers Collection

A **Containers** collection appears in each **Database** object of a Microsoft Jet database, and contains this method and this property.

Method

Refresh

Property

Count

Database Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjDatabaseP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjDatabaseM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjDatabaseS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjDatabaseU":1}

A **Database** object represents an open database.

TableDefs

Relations

Remarks

You use the **Database** object and its methods and properties to manipulate an open database. In any type of database, you can:

- Use the **Execute** method to run an action query.
- Set the **Connect** property to establish a connection to an ODBC data source.
- Set the **QueryTimeout** property to limit the length of time to wait for a query to execute against an ODBC data source.
- Use the **RecordsAffected** property to determine how many records were changed by an action query.
- Use the **OpenRecordset** method to execute a select query and create a **Recordset** object.
- Use the **Version** property to determine which version of a database engine created the database.

With a Microsoft Jet database (.mdb file), you can also use other methods, properties, and collections to manipulate a **Database** object, as well as create, modify, or get information about its tables, queries, and relationships. For example, you can:

- Use the **CreateTableDef** and **CreateRelation** methods to create tables and relations, respectively.
- Use the **CreateProperty** method to define new **Database** properties.
- Use the **CreateQueryDef** method to create a persistent or temporary query definition.
- Use **MakeReplica**, **Synchronize**, and **PopulatePartial** methods to create and synchronize full or partial replicas of your database.
- Set the **CollatingOrder** property to establish the alphabetic sorting order for character-based fields in different languages.

In an ODBCDirect workspace, you can:

- Use the **Connection** property to obtain a reference to the **Connection** object that corresponds to the **Database** object.

Note For a complete list of all methods, properties, and collections available on a **Database** object in either a Microsoft Jet workspace or ODBCDirect workspace, see the **Summary** topic.

You use the **CreateDatabase** method to create a persistent **Database** object that is automatically appended to the **Databases** collection, thereby saving it to disk.

You don't need to specify the **DBEngine** object when you use the **OpenDatabase** method.

Opening a database with linked tables doesn't automatically establish links to the specified external files or Microsoft Jet-connected ODBC data sources. You must either reference the table's **TableDef** or **Field** objects or open a **Recordset** object. If you can't establish links to these tables, a trappable error occurs. You may also need permission to access the database, or another user might have the database opened exclusively. In these cases, trappable errors occur.

You can also use the **OpenDatabase** method to open an external database (such as FoxPro, dBASE, and Paradox) directly instead of opening a Microsoft Jet database that has links to its tables.

Note Opening a **Database** object directly on a Microsoft Jet-connected ODBC data source, such as Microsoft SQL Server, is not recommended because query performance is much slower than when using linked tables. However, performance is not a problem with opening a **Database** object directly on an external ISAM database file, such as FoxPro, Paradox, and so forth.

When a procedure that declares a **Database** object has executed, local **Database** objects are closed along with any open **Recordset** objects. Any pending updates are lost and any pending transactions

are rolled back, but no trappable error occurs. You should explicitly complete any pending transactions or edits and close **Recordset** objects and **Database** objects before exiting procedures that declare these object variables locally.

When you use one of the transaction methods (**BeginTrans**, **CommitTrans**, or **Rollback**) on the **Workspace** object, these transactions apply to all databases opened on the **Workspace** from which the **Database** object was opened. If you want to use independent transactions, you must first open an additional **Workspace** object, and then open another **Database** object in that **Workspace** object.

Note You can open the same data source or database more than once, creating duplicate names in the **Databases** collection. You should assign **Database** objects to object variables and refer to them by variable name.

Databases Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolDatabaseP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolDatabaseM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolDatabaseS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolDatabaseU":1}

A **Databases** collection contains all open **Database** objects opened or created in a **Workspace** object.

TableDefs

Relations

Remarks

When you open an existing **Database** object or create a new one from a **Workspace**, it is automatically appended to the **Databases** collection. When you close a **Database** object with the **Close** method, it is removed from the **Databases** collection but not deleted from disk. You should close all open **Recordset** objects before closing a **Database** object.

In a Microsoft Jet workspace, the **Name** property setting of a database is a string that specifies the path of the database file. In an ODBCDirect workspace, the **Name** property is the name of the corresponding **Connection** object.

To refer to a **Database** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

Databases(0)
Databases("*name*")
Databases![*name*]

Note You can open the same data source or database more than once, creating duplicate names in the **Databases** collection. You should assign **Database** objects to object variables and refer to them by variable name.

Database Object, Databases Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumDatabaseS"}

Database Object

A **Database** object contains these collections, methods, and properties.

Legend:

Feature available in Microsoft Jet workspaces only.
DAO ODBC Feature available in ODBCDirect workspaces only.

Collections

Containers
Properties
QueryDefs
Recordsets (Default for DAO ODBC)

Relations

TableDefs (Default for)

Methods

Close

CreateProperty
CreateQueryDef

CreateRelation

CreateTableDef
Execute

MakeReplica

NewPassword
OpenRecordset
PopulatePartial

Synchronize

Properties

CollatingOrder
Connect

Connection

DesignMasterID
Name
QueryTimeout
RecordsAffected

Replicable (user-defined)

ReplicaID
Updatable

V1xNullBehavior
Version

Databases Collection

A **Databases** collection appears in each **Workspace** object, and contains this method and this property.

Method

Refresh

Property

Count

DBEngine Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjDBEngineC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjDBEngineX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjDBEngineP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjDBEngineM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjDBEngineS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjDBEngineU":1}

The **DBEngine** object is the top level object in the DAO object model.

Remarks

The **DBEngine** object contains and controls all other objects in the hierarchy of DAO objects. You can't create additional **DBEngine** objects, and the **DBEngine** object isn't an element of any collection.

Note When you reference an ODBC data source directly through DAO, it is called an "ODBCDirect workspace." This is to distinguish it from an ODBC data source that you reference indirectly through the Microsoft Jet database engine, using a "Microsoft Jet workspace." Each method of accessing ODBC data requires one of two types of **Workspace** object; you can set the **DefaultType** property to choose the default type of **Workspace** object that you will create from the **DBEngine** object. The **Workspace** type and associated data source determines which DAO objects, methods, and properties you can use.

With any type of database or connection, you can:

- Use the **Version** property to obtain the DAO version number.
- Use the **LoginTimeout** property to obtain or set the ODBC login timeout, and the **RegisterDatabase** method to provide ODBC information to the Microsoft Jet database engine. You can use these features the same way, regardless of whether you connect to the ODBC data source through Microsoft Jet or through an ODBCDirect workspace.
- Use the **DefaultType** property to set the default type of database connection that subsequently created **Workspace** objects will use — either Microsoft Jet or ODBCDirect.
- Use the **DefaultPassword** and **DefaultUser** properties to set the user identification and password for the default **Workspace** object.
- Use the **CreateWorkspace** method to create a new **Workspace** object. You can use optional arguments to override the settings of the **DefaultType**, **DefaultPassword**, and **DefaultUser** properties.
- Use the **OpenDatabase** method to open a database in the default **Workspace**, and use the **BeginTrans**, **Commit**, and **Rollback** methods to control transactions on the default **Workspace**.
- Use the **Workspaces** collection to reference specific **Workspace** objects.
- Use the **Errors** collection to examine data access error details.

Other properties and methods are only available when you use DAO with the Microsoft Jet database engine. You can use them to control the Microsoft Jet database engine, manipulate its properties, and

perform tasks on temporary objects that aren't elements of collections. For example, you can:

- Use the **CreateDatabase** method to create a new Microsoft Jet **Database** object.
- Use the **Idle** method to enable the Microsoft Jet database engine to complete any pending tasks.
- Use the **CompactDatabase** and **RepairDatabase** methods to maintain database files.
- Use the **IniPath** and **SystemDB** properties to specify the location of Microsoft Jet Windows Registry information and the Microsoft Jet workgroup information file, respectively. The **SetOption** method allows you override windows registry settings for the Microsoft Jet database engine.

After you change the **DefaultType** and **IniPath** property settings, only subsequent **Workspace** objects will reflect these changes.

Note For a complete list of all methods, properties, and collections available on the **DBEngine** object, see the **Summary** topic.

To refer to a collection that belongs to the **DBEngine** object, or to refer to a method or property that applies to this object, use this syntax:

[**DBEngine.**][*collection* **|** *method* **|** *property*]

DBEngine Object Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumDBEngineC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumDBEngineS"}

The **DBEngine** object contains these collections, methods, and properties.

Legend:

Feature available in Microsoft Jet workspaces only.

Feature available in ODBCDirect workspaces only.

Collections

Errors
Properties
Workspaces (Default)

Methods

BeginTrans
CommitTrans

CompactDatabase

CreateDatabase
CreateWorkspace

Idle

OpenConnection
OpenDatabase
RegisterDatabase

RepairDatabase
Rollback

SetOption

Properties

DefaultPassword
DefaultType
DefaultUser

IniPath
LoginTimeout

SystemDB
Version

Document Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjDocumentC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjDocumentX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjDocumentP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjDocumentM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjDocumentS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjDocumentU":1}

A **Document** object includes information about one instance of an object. The object can be a database, saved table, query, or relationship (Microsoft Jet databases only).

Remarks

Each **Container** object has a **Documents** collection containing **Document** objects that describe instances of built-in objects of the type specified by the **Container**. The following table lists the type of object each **Document** describes, the name of its **Container** object, and what type of information **Document** contains.

Document	Container	Contains information about
Database	Databases	Saved database
Table or query	Tables	Saved table or query
Relationship	Relations	Saved relationship

Note Don't confuse the **Container** objects listed in the preceding table with the collections of the same name. The Databases **Container** object refers to all saved database objects, but the **Databases** collection refers only to database objects that are open in a particular workspace.

With a **Document** object, you can:

- Use the **Name** property to return the name that a user or the Microsoft Jet database engine gave to the object when it was created.
- Use the **Container** property to return the name of the **Container** object that contains the **Document** object.
- Use the **Owner** property to set or return the owner of the object. To set the **Owner** property, you must have write permission for the **Document** object, and you must set the property to the name of an existing **User** or **Group** object.
- Use the **UserName** or **Permissions** properties to set or return the access permissions of a user or group for the object. To set these properties, you must have write permission for the **Document** object, and you must set the **UserName** property to the name of an existing **User** or **Group** object.
- Use the **DateCreated** and **LastUpdated** properties to return the date and time when the **Document** object was created and last modified.

Because a **Document** object corresponds to an existing object, you can't create new **Document** objects or delete existing ones. To refer to a **Document** object in a collection by its ordinal number or

by its **Name** property setting, use any of the following syntax forms:

Documents(0)
Documents("*name*")
Documents![*name*]

Documents Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolDocumentC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolDocumentX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolDocumentP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolDocumentM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolDocumentS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolDocumentU":1}

A **Documents** collection contains all of the **Document** objects for a specific type of object (Microsoft Jet databases only).

Remarks

Each **Container** object has a **Documents** collection containing **Document** objects that describe instances of built-in objects of the type specified by the **Container**.

To refer to a **Document** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

Documents(0)
Documents("*name*")
Documents![*name*]

Document Object, Documents Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumDocumentC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumDocumentS"}

Document Object

A **Document** object contains this collection, this method, and these properties.

Collection
Properties

Method
CreateProperty

Properties
AllPermissions
Container
DateCreated
KeepLocal (user-defined)
LastUpdated
Name
Owner
Permissions
Replicable (user-defined)
UserName

Documents Collection

A **Documents** collection appears in each **Container** object, and contains this method and this property.

Method
Refresh

Property
Count

Dynaset-Type Recordset Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjDynasetTypeRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjDynasetTypeRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjDynasetTypeRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjDynasetTypeRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjDynasetTypeRecordsetS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjDynasetTypeRecordsetU":1}

A dynaset-type **Recordset** object is a dynamic set of records that can contain fields from one or more tables or queries in a database and may be updatable. In an ODBCDirect database, a dynaset-type **Recordset** object corresponds to an ODBC keyset cursor.

Remarks

A dynaset-type **Recordset** object is a type of **Recordset** object you can use to manipulate data in an underlying database table or tables.

It differs from a snapshot-type **Recordset** object because the dynaset stores only the primary key for each record, instead of actual data. As a result, a dynaset is updated with changes made to the source data, while the snapshot is not. Like the table-type **Recordset** object, a dynaset retrieves the full record only when it's needed for editing or display purposes.

To create a dynaset-type **Recordset** object, use the **OpenRecordset** method on an open database, against another dynaset- or snapshot-type **Recordset** object, on a **QueryDef** object, or on a **TableDef** object. (Opening **Recordset** objects on other **Recordset** objects or **TableDef** objects is available only in Microsoft Jet workspaces.)

If you request a dynaset-type **Recordset** object and the Microsoft Jet database engine can't gain read/write access to the records, the Microsoft Jet database engine may create a read-only, dynaset-type **Recordset** object.

As users update data, the base tables reflects these changes. Therefore, current data is available to your application when you reposition the current record. In a multiuser database, more than one user can open a dynaset-type **Recordset** object referring to the same records. Because a dynaset-type **Recordset** object is dynamic, when one user changes a record, other users have immediate access to the changed data. However, if one user adds a record, other users won’t see the new record until they use the **Requery** method on the **Recordset** object. If a user deletes a record, other users are notified when they try to access it.

Records added to the database don't become a part of your dynaset-type **Recordset** object unless you add them by using the **AddNew** and **Update** methods. For example, if you use an action query containing an INSERT INTO SQL statement to add records, the new records aren't included in your dynaset-type **Recordset** object until you either use the **Requery** method or you rebuild your **Recordset** object using the **OpenRecordset** method.

To maintain data integrity, the Microsoft Jet database engine can lock dynaset- and table-type **Recordset** objects during **Edit** (pessimistic locking) or **Update** operations (optimistic locking) so that only one user can update a particular record at a time. When the Microsoft Jet database engine locks a record, it locks the entire 2K page containing the record.

You can also use optimistic and pessimistic locking with non-ODBC tables. When you access external tables using ODBC through a Microsoft Jet workspace, you should always use optimistic locking. The **LockEdits** property and the *lockedits* parameter of the **OpenRecordset** method determine the locking conditions during editing.

Not all fields can be updated in all dynaset-type **Recordset** objects. To determine whether you can update a particular field, check the **DataUpdatable** property setting of the **Field** object.

A dynaset-type **Recordset** object may not be updatable if:

- There isn't a unique index on the ODBC or Paradox table or tables.

- The data page is locked by another user.
- The record has changed since you last read it.
- The user doesn't have permission.
- One or more of the tables or fields are read-only.
- The database is opened as read-only.
- The **Recordset** object was either created from multiple tables without a JOIN statement or the query was too complex.

The order of a dynaset-type **Recordset** object or **Recordset** data doesn't necessarily follow any specific sequence. If you need to order your data, use an SQL statement with an ORDER BY clause to create the **Recordset** object. You can also use a WHERE clause to filter the records so that only certain records are added to the **Recordset** object. Using SQL statements in this way to select a subset of records and order them usually results in faster access to your data than using the **Filter** and **Sort** properties.

Error Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjErrorX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjErrorP"}
{ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjErrorM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjErrorS"}
{ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjErrorU":1}

An **Error** object contains details about data access errors, each of which pertains to a single operation involving DAO.

Remarks

Any operation involving DAO can generate one or more errors. For example, a call to an ODBC server might result in an error from the database server, an error from ODBC, and a DAO error. As each such error occurs, an **Error** object is placed in the **Errors** collection of the **DBEngine** object. A single event can therefore result in several **Error** objects appearing in the **Errors** collection.

When a subsequent DAO operation generates an error, the **Errors** collection is cleared, and one or more new **Error** objects are placed in the **Errors** collection. DAO operations that don't generate an error have no effect on the **Errors** collection.

The set of **Error** objects in the **Errors** collection describes one error. The first **Error** object is the lowest level error (the originating error), the second the next higher level error, and so forth. For example, if an ODBC error occurs while trying to open a **Recordset** object, the first **Error** object — **Errors**(0) — contains the lowest level ODBC error; subsequent errors contain the ODBC errors returned by the various layers of ODBC. In this case, the ODBC driver manager, and possibly the driver itself, return separate **Error** objects. The last **Error** object — **Errors.Count**-1 — contains the DAO error indicating that the object couldn't be opened.

Enumerating the specific errors in the **Errors** collection enables your error-handling routines to more precisely determine the cause and origin of an error, and take appropriate steps to recover. On both Microsoft Jet and ODBCDirect workspaces, you can read the **Error** object’s properties to obtain specific details about each error, including:

- The **Description** property, which contains the text of the error alert that will be displayed on the screen if the error is not trapped.
- The **Number** property, which contains the **Long** integer value of the error constant.
- The **Source** property, which identifies the object that raised the error. This is particularly useful when you have several **Error** objects in the **Errors** collection following a request to an ODBC data source.
- The **HelpFile** and **HelpContext** properties, which indicate the appropriate Microsoft Windows Help file and Help topic, respectively, (if any exist) for the error.

Note When programming in Microsoft Visual Basic for Applications (VBA), if you use the **New** keyword to create an object that subsequently causes an error before that object has been appended to a collection, the **DBEngine** object's **Errors** collection won't contain an entry for that object's error,

because the new object is not associated with the **DBEngine** object. However, the error information is available in the VBA **Err** object.

Your VBA error-handling code should examine the **Errors** collection whenever you anticipate a data access error. If you are writing a centralized error handler, test the VBA **Err** object to determine if the error information in the **Errors** collection is valid. If the **Number** property of the last element of the **Errors** collection (`DBEngine.Errors.Count - 1`) and the value of the **Err** object match, you can then use a series of **Select Case** statements to identify the particular DAO error or errors that occurred. If they do not match, use the **Refresh** method on the **Errors** collection.

Errors Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolErrorX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolErrorP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolErrorM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolErrorS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolErrorU":1}

An **Errors** collection contains all stored **Error** objects, each of which pertains to a single operation involving DAO.

Remarks

Any operation involving DAO objects can generate one or more errors. As each error occurs, one or more **Error** objects are placed in the **Errors** collection of the **DBEngine** object. When another DAO operation generates an error, the **Errors** collection is cleared, and the new set of **Error** objects is placed in the **Errors** collection. The highest-numbered object in the **Errors** collection (`DBEngine.Errors.Count - 1`) corresponds to the error reported by the Microsoft Visual Basic for Applications (VBA) **Err** object.

DAO operations that don't generate an error have no effect on the **Errors** collection.

Elements of the **Errors** collection aren't appended as they typically are with other collections, so the **Errors** collection doesn't support the **Append** and **Delete** methods.

The set of **Error** objects in the **Errors** collection describes one error. The first **Error** object is the lowest level error, the second the next higher level, and so forth. For example, if an ODBC error occurs while trying to open a **Recordset** object, the first error object contains the lowest level ODBC error; subsequent errors contain the ODBC errors returned by the various layers of ODBC. In this case, the ODBC driver manager, and possibly the driver itself, return separate **Error** objects. The last **Error** object contains the DAO error indicating that the object couldn't be opened.

Enumerating the specific errors in the **Errors** collection enables your error-handling routines to more precisely determine the cause and origin of an error, and take appropriate steps to recover.

Note If you use the **New** keyword to create an object that causes an error either before or while being placed into the **Errors** collection, the collection doesn't contain error information about that object, because the new object is not associated with the **DBEngine** object. However, the error information is available in the VBA **Err** object.

Error Object, Errors Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumErrorS"}

Error Object

An **Error** object contains no methods;no collections, and these properties:

Properties
Description
HelpContext
HelpFile
Number
Source

Errors Collection

An **Errors** collection appears in the **DBEngine** object, and contains this method and this property:

Method
Refresh

Property
Count

Field Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjFieldC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjFieldX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjFieldP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjFieldM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjFieldS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjFieldU":1}

A **Field** object represents a column of data with a common data type and a common set of properties.

Remarks

The **Fields** collections of **Index**, **QueryDef**, **Relation**, and **TableDef** objects contain the specifications for the fields those objects represent. The **Fields** collection of a **Recordset** object represents the **Field** objects in a row of data, or in a record. You use the **Field** objects in a **Recordset** object to read and set values for the fields in the current record of the **Recordset** object.

In both Microsoft Jet and ODBCDirect workspaces, you manipulate a field using a **Field** object and its methods and properties. For example, you can:

- Use the **OrdinalPosition** property to set or return the presentation order of the **Field** object in a **Fields** collection. (This property is read-only for ODBCDirect databases.)
- Use the **Value** property of a field in a **Recordset** object to set or return stored data.
- Use the **AppendChunk** and **GetChunk** methods and the **FieldSize** property to get or set a value in an OLE Object or Memo field of a **Recordset** object.
- Use the **Type**, **Size**, and **Attributes** properties to determine the type of data that can be stored in the field.
- Use the **SourceField** and **SourceTable** properties to determine the original source of the data.

In Microsoft Jet workspaces, you can:

- Use the **ForeignName** property to set or return information about a foreign field in a **Relation** object.
- Use the **AllowZeroLength**, **DefaultValue**, **Required**, **ValidateOnSet**, **ValidationRule**, or **ValidationText** properties to set or return validation conditions.
- Use the **DefaultValue** property of a field on a **TableDef** object to set the default value for this field when new records are added.

In ODBCDirect workspaces, you can:

- Use the **Value**, **VisibleValue**, and **OriginalValue** properties to verify successful completion of a batch update.

Note For a complete list of all methods, properties, and collections available on a **Field** object in any database or connection, see the Summary topic.

To create a new **Field** object in an **Index**, **TableDef**, or **Relation** object, use the **CreateField** method.

When you access a **Field** object as part of a **Recordset** object, data from the current record is visible in the **Field** object's **Value** property. To manipulate data in the **Recordset** object, you don't usually reference the **Fields** collection directly; instead, you indirectly reference the **Value** property of the **Field** object in the **Fields** collection of the **Recordset** object.

To refer to a **Field** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

Fields(0)
Fields("*name*")
Fields![*name*]

With the same syntax forms, you can also refer to the **Value** property of a **Field** object that you create and append to a **Fields** collection. The context of the field reference will determine whether you are referring to the **Field** object or the **Value** property of the **Field** object.

Fields Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolFieldC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolFieldX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolFieldP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolFieldM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolFieldS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolFieldU":1}

A **Fields** collection contains all stored **Field** objects of an **Index**, **QueryDef**, **Recordset**, **Relation**, or **TableDef** object.

Remarks

The **Fields** collections of the **Index**, **QueryDef**, **Relation**, and **TableDef** objects contain the specifications for the fields those objects represent. The **Fields** collection of a **Recordset** object represents the **Field** objects in a row of data, or in a record. You use the **Field** objects in a **Recordset** object to read and to set values for the fields in the current record of the **Recordset** object.

To refer to a **Field** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

Fields(0)
Fields("*name*")
Fields![*name*]

With the same syntax forms, you can also refer to the **Value** property of a **Field** object that you create and append to a **Fields** collection. The context of the field reference will determine whether you are referring to the **Field** object or the **Value** property of the **Field** object.

Field Object, Fields Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumFieldC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumFieldS"}

Field Object

A **Field** object contains this collection, these methods, and these properties.

Legend:

Feature available in Microsoft Jet workspaces only.

Feature available in ODBCDirect workspaces only.

Collection
Properties

Methods

The following table lists all of the **Field** object methods. The type of object whose **Fields** collection contains the **Field** object determines which methods are available.

Method	Index	QueryDef	Recordset	Relation	TableDef
AppendChunk			✔		
CreateProperty					
GetChunk			✔		

Properties

The following table lists all of the **Field** object properties. The type of object whose **Fields** collection contains the **Field** object determines which properties are available. All properties are read-only for **Field** objects appended to **Fields** collections of **Index**, **Relation,** and **TableDef** objects.

📖 Read-only
📝 Read/write

Property	Index	QueryDef	Recordset	Relation	TableDef
AllowZeroLength		📖	📖		📝
Attributes	📖	📖	📖		📖
CollatingOrder		📖	📖		
DataUpdatable	📖	📖	📖	📖	📖
DefaultValue		📖	📖		📝
FieldSize					
ForeignName					
Name					📝

Property		
OrdinalPosition		
OriginalValue	*	
Required		
Size		
SourceField		
SourceTable		
Type		
ValidateOnSet		
ValidationRule		
ValidationText		
Value		
VisibleValue	*	

* These properties are only available in an ODBCDirect workspace whose **DefaultCursorDriver** property is set to **dbUseClientBatchCursor.**

Fields Collection

A **Fields** collection appears in each of the **TableDef**, **QueryDef**, **Recordset**, **Relation**, and **Index** objects, and contains these methods and this property.

Method	Index	QueryDef	Recordset	Relation	TableDef
Append					
Delete					
Refresh					

Property	Index	QueryDef	Recordset	Relation	TableDef
Count					

Group Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjGroupC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjGroupX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjGroupP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjGroupM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjGroupS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjGroupU":1}

A **Group** object represents a group of user accounts that have common access permissions when a **Workspace** object operates as a secure workgroup. (Microsoft Jet workspaces only).

Remarks

You create **Group** objects and then use their names to establish and enforce access permissions for your databases, tables, and queries using the **Document** objects that represent the **Database**, **TableDef**, and **QueryDef** objects with which you're working.

With the properties of a **Group** object, you can:

- Use the **Name** property of an existing **Group** object to return its name. You can't return the **PID** property setting of an existing **Group** object.
- Use the **Name** and **PID** properties of a newly created, unappended **Group** object to set the identity of that **Group** object.

You can append an existing **Group** object to the **Groups** collection in a **User** object to establish membership of a user account in that **Group** object. Alternatively, you can append a **User** object to the **Users** collection in a **Group** object to give a user account the global permissions of that group. If you use a **Groups** or **Users** collection other than the one to which you just appended an object, you may need to use the **Refresh** method to refresh the collection with current information from the

database.

The Microsoft Jet database engine predefines three **Group** objects named Admins, Users, and Guests. To create a new **Group** object, use the **CreateGroup** method on a **User** or **Workspace** object.

To refer to a **Group** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

Groups(0)
Groups("*name*")
Groups![*name*]

Groups Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolGroupC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolGroupX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolGroupP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolGroupM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolGroupS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolGroupU":1}

A **Groups** collection contains all stored **Group** objects of a **Workspace** or user account (Microsoft Jet workspaces only).

Remarks

You can append an existing **Group** object to the **Groups** collection in a **User** object to establish membership of a user account in that **Group** object. Alternatively, you can append a **User** object to the **Users** collection in a **Group** object to give a user account the global permissions of that group. In either case, the existing **Group** object must already be a member of the **Groups** collection of the current **Workspace** object. If you use a **Groups** or **Users** collection other than the one to which you just appended an object, you may need to use the **Refresh** method to refresh the collection with current information from the database.

To refer to a **Group** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

Groups(0)
Groups("*name*")
Groups![*name*]

Group Object, Groups Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumGroupC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumGroupS"}

Group Object

A **Group** object contains these collections, this method, and these properties.

Collections

Properties
Users (default)

Method

CreateUser

Properties

Name
PID

Groups Collection

A **Groups** collection appears in each **User** and Microsoft Jet **Workspace** object, and contains these methods and this property.

Methods

Append
Delete
Refresh

Property

Count

Index Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjIndexX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjIndexP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjIndexM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjIndexS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjIndexU":1}

Index objects specify the order of records accessed from database tables and whether or not duplicate records are accepted, providing efficient access to data. For external databases, **Index** objects describe the indexes established for external tables (Microsoft Jet workspaces only).

Remarks

The Microsoft Jet database engine uses indexes when it joins tables and creates **Recordset** objects. Indexes determine the order in which table-type **Recordset** objects return records, but they don't determine the order in which the Microsoft Jet database engine stores records in the base table or the order in which any other type of **Recordset** object returns records.

With an **Index** object, you can:

- Use the **Required** property to determine whether the **Field** objects in the index require values that are not **Null**, and then use the **IgnoreNulls** property to determine whether the **Null** values have index entries.
- Use the **Primary** and **Unique** properties to determine the ordering and uniqueness of the **Index** object.

The Microsoft Jet database engine maintains all base table indexes automatically. It updates indexes whenever you add, change, or delete records from the base table. Once you create the database, use the **CompactDatabase** method periodically to bring index statistics up-to-date.

When accessing a table-type **Recordset** object, you specify the order of records using the object's **Index** property. Set this property to the **Name** property setting of an existing **Index** object in the **Indexes** collection. This collection is contained by the **TableDef** object underlying the **Recordset** object that you're populating.

Note You don't have to create indexes for a table, but for large, unindexed tables, accessing a specific record or processing joins can take a long time. Conversely, having too many indexes can slow down updates to the database as each of the table indexes is amended.

The **Attributes** property of each **Field** object in the index determines the order of records returned and consequently determines which access techniques to use for that index.

Each **Field** object in the **Fields** collection of an **Index** object is a component of the index. To define a new **Index** object, set its properties before you append it to a collection, making the **Index** object available for subsequent use.

Note You can modify the **Name** property setting of an existing **Index** object only if the **Updatable** property setting of the containing **TableDef** object is **True**.

When you set a primary key for a table, the Microsoft Jet database engine automatically defines it as the primary index. A primary index consists of one or more fields that uniquely identify all records in a table in a predefined order. Because the primary index field must be unique, the Microsoft Jet database engine automatically sets the **Unique** property of the primary **Index** object to **True**. If the primary index consists of more than one field, each field can contain duplicate values, but the combination of values from all the indexed fields must be unique. A primary index consists of a key for the table and is always made up of the same fields as the primary key.

Important Make sure your data complies with the attributes of your new index. If your index requires unique values, make sure that there are no duplicates in existing data records. If duplicates exist, the Microsoft Jet database engine can't create the index; a trappable error results when you attempt to use the **Append** method on the new index.

When you create a relationship that enforces referential integrity, the Microsoft Jet database engine automatically creates an index with the **Foreign** property, set as the foreign key in the referencing table. After you've established a table relationship, the Microsoft Jet database engine prevents additions or changes to the database that violate that relationship. If you set the **Attributes** property of the **Relation** object to allow cascading updates and cascading deletes, the Microsoft Jet database engine updates or deletes records in related tables automatically.

To create a new Index object

1. Use the **CreateIndex** method on a **TableDef** object.
2. Use the **CreateField** method on the **Index** object to create a **Field** object for each field (column) to be included in the **Index** object.
3. Set **Index** properties as needed.
4. Append the **Field** object to the **Fields** collection.
5. Append the **Index** object to the **Indexes** collection.

Note The **Clustered** property is ignored for databases that use the Microsoft Jet database engine, which doesn't support clustered indexes.

Indexes Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolIndexX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolIndexP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolIndexM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolIndexS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolIndexU":1}

An **Indexes** collection contains all the stored **Index** objects of a **TableDef** object (Microsoft Jet workspaces only).

Remarks

When you access a table-type **Recordset** object, use the object's **Index** property to specify the order of records. Set this property to the **Name** property setting of an existing **Index** object in the **Indexes** collection of the the **TableDef** object underlying the **Recordset** object.

Note You can use the **Append** or **Delete** method on an **Indexes** collection only if the **Updatable** property setting of the containing **TableDef** object is **True**.

After you create a new **Index** object, you should use the **Append** method to add it to the **TableDef** object's **Indexes** collection.

Important Make sure your data complies with the attributes of your new index. If your index requires unique values, make sure that there are no duplicates in existing data records. If duplicates exist, the Microsoft Jet database engine can't create the index; a trappable error results when you attempt to use the **Append** method on the new index.

Index Object, Indexes Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumIndexS"}

Index Object

An **Index** object contains these collections, methods, and properties.

Collections

Fields (default)
Properties

Methods

CreateField
CreateProperty

Properties

Clustered
DistinctCount
Foreign
IgnoreNulls
Name
Primary
Required
Unique

Indexes Collection

An **Indexes** collection appears in each **TableDef** object, and contains these methods and this property.

Methods

Append
Delete
Refresh

Property

Count

Parameter Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjParameterC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjParameterX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjParameterP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjParameterM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjParameterS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjParameterU":1}

A **Parameter** object represents a value supplied to a query. The parameter is associated with a **QueryDef** object created from a parameter query.

Remarks

Parameter objects allow you to change the arguments in a frequently run **QueryDef** object without having to recompile the query.

Using the properties of a **Parameter** object, you can set a query parameter that can be changed before the query is run. You can:

- Use the **Name** property to return the name of a parameter.
- Use the **Value** property to set or return the parameter values to be used in the query.
- Use the **Type** property to return the data type of the **Parameter** object.
- Use the **Direction** property to set or return whether the parameter is an input parameter, an output parameter, or both.

In an ODBCDirect workspace, you can also:

- Change the setting of the **Type** property. Doing so will also clear the Value property.
- Use the **Direction** property to set or return whether the parameter is an input parameter, an output parameter, or both.

Parameters Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolParameterC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolParameterX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolParameterP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolParameterM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolParameterS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolParameterU":1}

A **Parameters** collection contains all the **Parameter** objects of a **QueryDef** object.

Remarks

The **Parameters** collection provides information only about existing parameters. You can't append objects to or delete objects from the **Parameters** collection.

Parameter Object, Parameters Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumParameterC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumParameterS"}

Parameter Object

A **Parameter** object contains no methods; it contains this collection and these properties.

Legend:

Feature available in ODBCDirect workspaces only.

Collection
Properties

Properties
Direction
Name
Type
Value (Default)

Parameters Collection

A **Parameters** collection appears in each **QueryDef** object and contains this method and this property.

Method
Refresh

Property
Count

Property Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjPropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjPropertyX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjPropertyP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjPropertyM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjPropertyS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjPropertyU":1}

A **Property** object represents a built-in or user-defined characteristic of a DAO object.

Property

Remarks

Every DAO object except the **Connection** and **Error** objects contains a **Properties** collection which has **Property** objects corresponding to built-in properties of that DAO object. The user can also define **Property** objects and append them to the **Properties** collection of some DAO objects. These **Property** objects (which are often just called properties) uniquely characterize that instance of the object.

You can create user-defined properties for the following objects:

- **Database**, **Index**, **QueryDef**, and **TableDef** objects
- **Field** objects in **Fields** collections of **QueryDef** and **TableDef** objects

To add a user-defined property, use the **CreateProperty** method to create a **Property** object with a unique **Name** property setting. Set the **Type** and **Value** properties of the new **Property** object, and then append it to the **Properties** collection of the appropriate object. The object to which you are adding the user-defined property must already be appended to a collection. Referencing a user-defined **Property** object that has not yet been appended to a **Properties** collection will cause an error, as will appending a user-defined **Property** object to a **Properties** collection containing a **Property** object of the same name.

You can delete user-defined properties from the **Properties** collection, but you can't delete built-in properties.

Note A user-defined **Property** object is associated only with the specific instance of an object. The property isn't defined for all instances of objects of the selected type.

You can use the **Properties** collection of an object to enumerate the object's built-in and user-defined properties. You don't need to know beforehand exactly which properties exist or what their characteristics (**Name** and **Type** properties) are to manipulate them. However, if you try to read a write-only property, such as the **Password** property of a **Workspace** object, or try to read or write a property in an inappropriate context, such as the **Value** property setting of a **Field** object in the **Fields** collection of a **TableDef** object, an error occurs.

The **Property** object also has four built-in properties:

- The **Name** property, a **String** that uniquely identifies the property.
- The **Type** property, an **Integer** that specifies the property data type.
- The **Value** property, a **Variant** that contains the property setting.

- The **Inherited** property, a **Boolean** that indicates whether the property is inherited from another object. For example, a **Field** object in a **Fields** collection of a **Recordset** object can inherit properties from the underlying **TableDef** or **QueryDef** object.

To refer to a built-in **Property** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

*object***.Properties**(0)
*object***.Properties**("*name*")
*object***.Properties**![*name*]

For a built-in property, you can also use this syntax:

object.name

Note For a user-defined property, you must use the full *object***.Properties**("*name*") syntax.

With the same syntax forms, you can also refer to the **Value** property of a **Property** object. The context of the reference will determine whether you are referring to the **Property** object itself or the **Value** property of the **Property** object.

Properties Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolPropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolPropertyX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolPropertyP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolPropertyM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolPropertyS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolPropertyU":1}

A **Properties** collection contains all the **Property** objects for a specific instance of an object.

All DAO Objects

Properties

Property

Remarks

Every DAO object except the **Connection** and **Error** objects contains a **Properties** collection, which has certain built-in **Property** objects. These **Property** objects (which are often just called properties) uniquely characterize that instance of the object.

In addition to the built-in properties, you can also create and add your own user-defined properties. To add a user-defined property to an existing instance of an object, first define its characteristics with the **CreateProperty** method, then add it to the collection with the **Append** method. Referencing a user-defined **Property** object that has not yet been appended to a **Properties** collection will cause an error, as will appending a user-defined **Property** object to a **Properties** collection containing a **Property** object of the same name.

You can use the **Delete** method to remove user-defined properties from the **Properties** collection, but you can't remove built-in properties.

Note A user-defined **Property** object is associated only with the specific instance of an object. The property isn't defined for all instances of objects of the selected type.

You can use the **Properties** collection of an object to enumerate the object's built-in and user-defined properties. You don't need to know beforehand exactly which properties exist or what their characteristics (**Name** and **Type** properties) are to manipulate them. However, if you try to read a write-only property, such as the **Password** property of a **Workspace** object, or try to read or write a property in an inappropriate context, such as the **Value** property setting of a **Field** object in the **Fields** collection of a **TableDef** object, an error occurs.

To refer to a built-in **Property** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

object.**Properties**(0)
object.**Properties**("*name*")
object.**Properties**![*name*]

For a built-in property, you can also use this syntax:

object.name

Note For a user-defined property, you must use the full *object*.**Properties**("*name*") syntax.

With the same syntax forms, you can also refer to the **Value** property of a **Property** object. The

context of the reference will determine whether you are referring to the **Property** object itself or the **Value** property of the **Property** object.

Property Object, Properties Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumPropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumPropertyS"}

Property Object

A **Property** object contains no methods; it contains this collection and these properties.

Collection
Properties

Properties
Inherited (Always False in ODBCDirect databases)
Name
Type
Value

Properties Collection

A **Properties** collection appears in each of the other DAO objects except the **Connection** and **Error** objects, and contains these methods and this property.

Methods
Append
Delete
Refresh

Property
Count

QueryDef Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjQueryDefC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjQueryDefX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjQueryDefP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjQueryDefM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjQueryDefS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjQueryDefU":1}

A **QueryDef** object is a stored definition of a query in a Microsoft Jet database, or a temporary definition of a query in an ODBCDirect workspace.

Connection

Remarks

You can use the **QueryDef** object to define a query. For example, you can:

- Use the **SQL** property to set or return the query definition.
- Use the **QueryDef** object's **Parameters** collection to set or return query parameters.

- Use the **Type** property to return a value indicating whether the query selects records from an existing table, makes a new table, inserts records from one table into another table, deletes records, or updates records.
- Use the **MaxRecords** property to limit the number of records returned from a query.
- Use the **ODBCTimeout** property to indicate how long to wait before the query returns records. The **ODBCTimeout** property applies to any query that accesses ODBC data.

In a Microsoft Jet workspace, you can also:

- Use the **ReturnsRecords** property to indicate that the query returns records. The **ReturnsRecords** property is only valid on SQL pass-through queries.
- Use the **Connect** property to make an SQL pass-through query to an ODC database.

In an ODBCDirect workspace, you can also:

- Use the **Prepare** property to determine whether to invoke the ODBC **SQLPrepare** API when the query is executed.
- Use the **CacheSize** property to cache records returned from a query.

You can also create temporary **QueryDef** objects. Unlike permanent **QueryDef** objects, temporary **QueryDef** objects are not saved to disk or appended to the **QueryDefs** collection. Temporary **QueryDef** objects are useful for queries that you must run repeatedly during run time but do not not need to save to disk, particularly if you create their SQL statements during run time.

You can think of a permanent **QueryDef** object in a Microsoft Jet workspaces as a compiled SQL statement. If you execute a query from a permanent **QueryDef** object, the query will run faster than if you run the equivalent SQL statement from the **OpenRecordset** method. This is because the Microsoft Jet database engine doesn't need to compile the query before executing it.

The preferred way to use the native SQL dialect of an external database engine accessed through the Microsoft Jet database engine is through **QueryDef** objects. For example, you can create a Microsoft SQL Server query and store it in a **QueryDef** object. When you need to use a non-Microsoft Jet database engine SQL query, you must provide a **Connect** property string that points to the external data source. Queries with valid **Connect** properties bypass the Microsoft Jet database engine and pass the query directly to the external database server for processing.

To create a new **QueryDef** object, use the **CreateQueryDef** method. In a Microsoft Jet workspace, if you supply a string for the *name* argument or if you explicitly set the **Name** property of the new **QueryDef** object to a non–zero-length string, you will create a permanent **QueryDef** that will automatically be appended to the **QueryDefs** collection and saved to disk. Supplying a zero-length string as the *name* argument or explicitly setting the **Name** property to a zero-length string will result in a temporary **QueryDef** object.

In an ODBCDirect workspace, a **QueryDef** is always temporary. The **QueryDefs** collection contains all open **QueryDef** objects. When a **QueryDef** is closed, it is automatically removed from the **QueryDefs** collection.

To refer to a **QueryDef** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

QueryDefs(0)
QueryDefs("*name*")
QueryDefs![*name*]

You can refer to temporary **QueryDef** objects only by the object variables that you have assigned to them.

QueryDefs Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolQueryDefC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolQueryDefX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolQueryDefP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolQueryDefM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolQueryDefS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolQueryDefU":1}

A **QueryDefs** collection contains all **QueryDef** objects of a **Database** object in a Microsoft Jet database, and all **QueryDef** objects of a **Connection** object in an ODBCDirect workspace.

Connection

Remarks

To create a new **QueryDef** object, use the **CreateQueryDef** method. In a Microsoft Jet workspace, if you supply a string for the *name* argument or if you explicitly set the **Name** property of the new **QueryDef** object to a non–zero-length string, you will create a permanent **QueryDef** that will automatically be appended to the **QueryDefs** collection and saved to disk. Supplying a zero-length

string as the *name* argument or explicitly setting the **Name** property to a zero-length string will result in a temporary **QueryDef** object.

In an ODBCDirect workspace, a **QueryDef** is always temporary. The **QueryDefs** collection contains all open **QueryDef** objects. When a **QueryDef** is closed, it is automatically removed from the **QueryDefs** collection.

To refer to a **QueryDef** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

QueryDefs(0)
QueryDefs("*name*")
QueryDefs![*name*]

You can refer to temporary **QueryDef** objects only by the object variables that you have assigned to them.

QueryDef Object, QueryDefs Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumQueryDefC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumQueryDefS"}

QueryDef Object

A **QueryDef** object contains these collections, methods, and properties.

Legend:

Available only in a Microsoft Jet workspace.

Available only in an ODBCDirect workspace.

Collections

Fields
Parameters (default)
Properties

Methods

Cancel
Close

CreateProperty
Execute
OpenRecordset

Properties

CacheSize
Connect

DateCreated

KeepLocal

LastUpdated

LogMessages
MaxRecords
Name
ODBCTimeout

Prepare
RecordsAffected

Replicable

ReturnsRecords
SQL

StillExecuting
Type
Updatable

QueryDefs Collection

A **QueryDefs** collection appears in each **Connection** object in an ODBCDirect workspace, and each **Database** object, and contains these methods and this property.

Methods

Append
Delete
Refresh

Property

Count

Recordset Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjRecordsetS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjRecordsetU":1}

A **Recordset** object represents the records in a base table or the records that result from running a query.

Connection

Remarks

You use **Recordset** objects to manipulate data in a database at the record level. When you use DAO objects, you manipulate data almost entirely using **Recordset** objects. All **Recordset** objects are constructed using records (rows) and fields (columns). There are five types of **Recordset** objects:

- **Table-type Recordset** — representation in code of a base table that you can use to add, change, or delete records from a single database table (Microsoft Jet workspaces only).
- **Dynaset-type Recordset** — the result of a query that can have updatable records. A dynaset-type **Recordset** object is a dynamic set of records that you can use to add, change, or delete records from an underlying database table or tables. A dynaset-type **Recordset** object can contain fields from one or more tables in a database. This type corresponds to an ODBC keyset cursor.
- **Snapshot-type Recordset** — a static copy of a set of records that you can use to find data or generate reports. A snapshot-type **Recordset** object can contain fields from one or more tables in a database but can't be updated. This type corresponds to an ODBC static cursor.
- **Forward-only-type Recordset** — identical to a snapshot except that no cursor is provided. You

can only scroll forward through records. This improves performance in situations where you only need to make a single pass through a result set. This type corresponds to an ODBC forward-only cursor.

- **Dynamic-type Recordset** — a query result set from one or more base tables in which you can add, change, or delete records from a row-returning query. Further, records other users add, delete, or edit in the base tables also appear in your **Recordset**. This type corresponds to an ODBC dynamic cursor (ODBCDirect workspaces only).

You can choose the type of **Recordset** object you want to create using the *type* argument of the **OpenRecordset** method.

In a Microsoft Jet workspace, if you don't specify a *type*, DAO attempts to create the type of **Recordset** with the most functionality available, starting with table. If this type isn't available, DAO attempts a dynaset, then a snapshot, and finally a forward-only type **Recordset** object.

In an ODBCDirect workspace, if you don't specify a *type*, DAO attempts to create the type of **Recordset** with the fastest query response, starting with forward-only. If this type isn't available, DAO attempts a snapshot, then a dynaset, and finally a dynamic- type **Recordset** object.

When creating a **Recordset** object using a non-linked **TableDef** object in a Microsoft Jet workspace, table-type **Recordset** objects are created. Only dynaset-type or snapshot-type **Recordset** objects can be created with linked tables or tables in Microsoft Jet-connected ODBC databases.

A new **Recordset** object is automatically added to the **Recordsets** collection when you open the object, and is automatically removed when you close it.

Note If you use variables to represent a **Recordset** object and the **Database** object that contains the **Recordset**, make sure the variables have the same scope, or lifetime. For example, if you declare a public variable that represents a **Recordset** object, make sure the variable that represents the **Database** containing the **Recordset** is also public, or is declared in a **Sub** or **Function** procedure using the **Static** keyword.

You can create as many **Recordset** object variables as needed. Different **Recordset** objects can access the same tables, queries, and fields without conflicting.

Dynaset-, snapshot-, and forward-only–type **Recordset** objects are stored in local memory. If there isn't enough space in local memory to store the data, the Microsoft Jet database engine saves the additional data to TEMP disk space. If this space is exhausted, a trappable error occurs.

The default collection of a **Recordset** object is the **Fields** collection, and the default property of a **Field** object is the **Value** property. Use these defaults to simplify your code.

When you create a **Recordset** object, the current record is positioned to the first record if there are any records. If there are no records, the **RecordCount** property setting is 0, and the **BOF** and **EOF** property settings are **True**.

You can use the **MoveNext**, **MovePrevious**, **MoveFirst**, and **MoveLast** methods to reposition the current record. Forward-only–type **Recordset** objects support only the **MoveNext** method. When using the Move methods to visit each record (or "walk" through the **Recordset**), you can use the **BOF** and **EOF** properties to check for the beginning or end of the **Recordset** object.

With dynaset- and snapshot-type **Recordset** objects in a Microsoft Jet workspace, you can also use the Find methods, such as **FindFirst**, to locate a specific record based on criteria. If the record isn't found, the **NoMatch** property is set to **True**. For table-type **Recordset** objects, you can scan records using the **Seek** method.

The **Type** property indicates the type of **Recordset** object created, and the **Updatable** property indicates whether you can change the object's records.

Information about the structure of a base table, such as the names and data types of each **Field** object and any **Index** objects, is stored in a **TableDef** object.

To refer to a **Recordset** object in a collection by its ordinal number or by its **Name** property setting,

use any of the following syntax forms:

Recordsets(0)
Recordsets("*name*")
Recordsets![*name*]

Note You can open a **Recordset** object from the same data source or database more than once, creating duplicate names in the **Recordsets** collection. You should assign **Recordset** objects to object variables and refer to them by variable name.

Recordsets Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolRecordsetS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolRecordsetU":1}

A **Recordsets** collection contains all open **Recordset** objects in a **Database** object.

Connection

Remarks

When you use DAO objects, you manipulate data almost entirely using **Recordset** objects.

A new **Recordset** object is automatically added to the **Recordsets** collection when you open the **Recordset** object, and is automatically removed when you close it.

You can create as many **Recordset** object variables as needed. Different **Recordset** objects can access the same tables, queries, and fields without conflicting.

To refer to a **Recordset** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

Recordsets(0)
Recordsets("*name*")
Recordsets![*name*]

Note You can open a **Recordset** object from the same data source or database more than once, creating duplicate names in the **Recordsets** collection. You should assign **Recordset** objects to object variables and refer to them by variable name.

Recordset Object, Recordsets Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumRecordsetS"}

Recordset Object

A **Recordset** object contains these collections, methods, and properties.

Legend:

Feature available in both Microsoft Jet and ODBCDirect workspaces.

Feature available in Microsoft Jet workspaces only.

Feature available in ODBCDirect workspaces only.

Collections
Fields (default)
Properties

Recordset Methods

The following table lists all of the **Recordset** methods, and shows which **Recordset** type supports each method, and whether the method is available in either a Microsoft Jet or ODBCDirect workspace, or both.

Method	Table	Dynaset	Snapshot	Forward-Only	Dynamic
AddNew			*		
Cancel					
CancelUpdate			*		
Clone					
Close					
CopyQueryDef					
Delete			*		
Edit			*		
FillCache					
FindFirst					
FindLast					
FindNext					
FindPrevious					
GetRows					

Method					
Move				Only with forward moves that don't use a bookmark offset	
MoveFirst					
MoveLast					
MoveNext					
MovePrevious					
NextRecordset					
OpenRecordset					
Requery					
Seek					
Update					

*

* In an ODBCDirect workspace, a snapshot-type **Recordset** may be updatable, depending on the ODBC driver. The **AddNew**, **Edit**, **Delete**, **Update**, and **CancelUpdate** methods are only available on ODBCDirect snapshot-type **Recordset** objects if the ODBC driver supports updatable snapshots.

Recordset Properties

The following table indicates which properties apply to each type of **Recordset** object and whether the property setting is read/write, read-only, or always **False** in either Microsoft Jet or ODBCDirect databases.

Read-only

Read/write

Property	Table	Dynaset	Snapshot	Forward-Only	Dynamic
AbsolutePosition					
BatchCollisionCount					
BatchCollisions					
BatchSize					
BOF					

Bookmark		
Bookmarkable		
CacheSize	for Microsoft Jet workspaces for ODBCDirect workspaces	
CacheStart		
Connection		
DateCreated		
EditMode		
EOF		
Filter		
Index		
LastModified		*
LastUpdated		
LockEdits	for Microsoft Jet workspaces for ODBCDirect workspaces	for Microsoft Jet workspaces for ODBCDirect workspaces
Name		

NoMatch				
PercentPosition				
RecordCount				
RecordStatus				
Restartable	**False**			
Sort				
StillExecuting				
Transactions			Always **False**	Always **False**
Type				
Updatable			Always **False** in Microsoft Jet workspaces in ODBCDirect workspaces *	Always **False** in Microsoft Jet workspaces in ODBCDirect workspaces *
UpdateOptions				
ValidationRule				
ValidationText				

*In an ODBCDirect workspace, a snapshot-type **Recordset** may be updatable, depending on the ODBC driver. The **LastModified** property is available, and the **Updatable** property is **True** only on ODBCDirect snapshot-type **Recordset** objects if the ODBC driver supports updatable snapshots.

Recordsets Collection

A **Recordsets** collection appears in each **Connection** and **Database** object, and contains this method and this property.

Method
Refresh

Property
Count

Relation Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjRelationC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjRelationX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjRelationP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjRelationM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjRelationS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjRelationU":1}

A **Relation** object represents a relationship between fields in tables or queries (Microsoft Jet databases only).

Remarks

You can use the **Relation** object to create new relationships and examine existing relationships in your database.

Using a **Relation** object and its properties, you can:

- Specify an enforced relationship between fields in base tables (but not a relationship that involves a query or a linked table).
- Establish unenforced relationships between any type of table or query — native or linked.
- Use the **Name** property to refer to the relationship between the fields in the referenced primary table and the referencing foreign table.
- Use the **Attributes** property to determine whether the relationship between fields in the table is one-to-one or one-to-many and how to enforce referential integrity.
- Use the **Attributes** property to determine whether the Microsoft Jet database engine can perform cascading update and cascading delete operations on primary and foreign tables.
- Use the **Attributes** property to determine whether the relationship between fields in the table is left join or right join.
- Use the **Name** property of all **Field** objects in the **Fields** collection of a **Relation** object to set or return the names of the fields in the primary key of the referenced table, or the **ForeignName** property settings of the **Field** objects to set or return the names of the fields in the foreign key of the referencing table.

If you make changes that violate the relationships established for the database, a trappable error occurs. If you request cascading update or cascading delete operations, the Microsoft Jet database engine also modifies the primary or foreign key tables to enforce the relationships you establish.

For example, the Northwind database contains a relationship between an Orders table and a Customers table. The CustomerID field of the Customers table is the primary key, and the CustomerID field of the Orders table is the foreign key. For Microsoft Jet to accept a new record in the Orders table, it searches the Customers table for a match on the CustomerID field of the Orders table. If Microsoft Jet doesn't find a match, it doesn't accept the new record, and a trappable error occurs.

When you enforce referential integrity, a unique index must already exist for the key field of the referenced table. The Microsoft Jet database engine automatically creates an index with the **Foreign** property set to act as the foreign key in the referencing table.

To create a new **Relation** object, use the **CreateRelation** method. To refer to a **Relation** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

Relations(0)
Relations("*name*")
Relations![*name*]

Relations Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolRelationC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolRelationX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolRelationP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolRelationM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolRelationS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolRelationU":1}

A **Relations** collection contains stored **Relation** objects of a **Database** object (Microsoft Jet databases only).

Remarks

You can use the **Relation** object to create new relationships and examine existing relationships in your database. To add a **Relation** object to the **Relations** collection, first create it with the **CreateRelation** method, and then append it to the **Relations** collection with the **Append** method. This will save the **Relation** object when you close the **Database** object. To remove a **Relation** object from the collection, use the **Delete** method.

To refer to a **Relation** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

Relations(0)
Relations("*name*")
Relations![*name*]

Relation Object, Relations Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumRelationC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumRelationS"}

Relation Object

A **Relation** object contains these collections, this method, and these properties.

Collections

Fields (Default)

Properties

Method

CreateField

Properties

Attributes

ForeignTable

Name

PartialReplica

Table

Relations Collection

A **Relations** collection is contained in each **Database** object of a Microsoft Jet database, and contains these methods and this property.

Methods

Append

Delete

Refresh

Property

Count

Snapshot-Type Recordset Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjSnapshotTypeRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjSnapshotTypeRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjSnapshotTypeRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjSnapshotTypeRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjSnapshotTypeRecordsetS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjSnapshotTypeRecordsetU":1}

A snapshot-type **Recordset** object is a static set of records that you can use to examine data in an underlying table or tables. In an ODBCDirect database, a snapshot-type **Recordset** object corresponds to a static cursor.

Remarks

To create a snapshot-type **Recordset** object, use the **OpenRecordset** method on an open database, on another dynaset- or snapshot-type **Recordset** object, or on a **QueryDef** object.

A snapshot-type **Recordset** object can contain fields from one or more tables in a database. In a Microsoft Jet workspace, a snapshot can't be updated. In an ODBCDirect workspace, a snapshot may be updatable, depending on the ODBC driver.

When you create a snapshot-type **Recordset** object, data values for all fields (except Memo and OLE Object (Long Binary) field data types in .mdb files) are brought into memory. Once loaded, changes made to base table data aren't reflected in the snapshot-type **Recordset** object data. To reload the snapshot-type **Recordset** object with current data, use the **Requery** method, or re-execute the **OpenRecordset** method.

The order of snapshot-type **Recordset** object data doesn't necessarily follow any specific sequence. To order your data, use an SQL statement with an ORDER BY clause to create the **Recordset** object. You can also use this technique to filter the records so that only certain records are added to the **Recordset** object. Using this technique instead of using the **Filter** or **Sort** properties or testing each record individually generally results in faster access to your data.

Snapshot-type **Recordset** objects are generally faster to create and access than dynaset-type **Recordset** objects because their records are either in memory or stored in TEMP disk space, and the Microsoft Jet database engine doesn't need to lock pages or handle multiuser issues. However, snapshot-type **Recordset** objects use more resources than dynaset-type **Recordset** objects because the entire record is downloaded to local memory.

Table-Type Recordset Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjTableTypeRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjTableTypeRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjTableTypeRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjTableTypeRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjTableTypeRecordsetS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjTableTypeRecordsetU":1}

A table-type **Recordset** object represents a base table you can use to add, change, or delete records from a table. Only the current record is loaded into memory. A predefined index determines the order of the records in the **Recordset** object (Microsoft Jet workspaces only).

Remarks

To create a table-type **Recordset** object, use the **OpenRecordset** method on an open **Database** object.

You can create a table-type **Recordset** object from a base table of a Microsoft Jet database, but not from an ODBC or linked table. You can use the table-type **Recordset** object with ISAM databases (like FoxPro, dBASE, or Paradox) when you open them directly.

Unlike dynaset- or snapshot-type **Recordset** objects, the table-type **Recordset** object can't refer to more than one base table, and you can't create it with an SQL statement that filters or sorts the data. Generally, when you access a table-type **Recordset** object, you specify one of the predefined indexes for the table, which orders the data returned to your application. If the table doesn't have an index, the data won't necessarily be in a particular order. If necessary, your application can create an index that returns records in a specific order. To choose a specific order for your table-type **Recordset** object, set the **Index** property to a valid index.

Also unlike dynaset- or snapshot-type **Recordset** objects, you don't need to explicitly populate table-type **Recordset** objects to obtain an accurate value for the **RecordCount** property.

To maintain data integrity, table-type **Recordset** objects are locked during the **Edit** and **Update** methods operations so that only one user can update a particular record at a time. When the Microsoft Jet database engine locks a record, it locks the entire 2K page containing the record.

Two kinds of locking are used with non-ODBC tables — pessimistic and optimistic. ODBC-accessed tables always use optimistic locking. The **LockEdits** property determines the locking conditions in effect during editing.

TableDef Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjTableDefC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjTableDefX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjTableDefP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjTableDefM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjTableDefS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjTableDefU":1}

A **TableDef** object represents the stored definition of a base table or a linked table (Microsoft Jet workspaces only).

Remarks

You manipulate a table definition using a **TableDef** object and its methods and properties. For example, you can:

- Examine the field and index structure of any local, linked, or external table in a database.
- Use the **Connect** and **SourceTableName** properties to set or return information about linked tables, and use the **RefreshLink** method to update connections to linked tables.
- Use the **ValidationRule** and **ValidationText** properties to set or return validation conditions.
- Use the **OpenRecordset** method to create a table-, dynaset-, dynamic-, snapshot-, or forward-only–type **Recordset** object, based on the table definition.

For base tables, the **RecordCount** property contains the number of records in the specified database table. For linked tables, the **RecordCount** property setting is always -1.

To create a new **TableDef** object, use the **CreateTableDef** method.

To add a field to a table

1. Make sure any **Recordset** objects based on the table are all closed.
2. Use the **CreateField** method to create a **Field** object variable and set its properties.
1. Use the **Append** method to add the **Field** object to the **Fields** collection of the **TableDef** object.

You can delete a **Field** object from a **TableDefs** collection if it doesn't have any indexes assigned to it, but you will lose the field's data.

To create a table that is ready for new records in a database

1. Use the **CreateTableDef** method to create a **TableDef** object.
1. Set its properties.
1. For each field in the table, use the **CreateField** method to create a **Field** object variable and set its properties.
2. Use the **Append** method to add the fields to the **Fields** collection of the **TableDef** object.
3. Use the **Append** method to add the new **TableDef** object to the **TableDefs** collection of the **Database** object.

A linked table is connected to the database by the **SourceTableName** and **Connect** properties of the **TableDef** object.

To link a table to a database

1. Use the **CreateTableDef** method to create a **TableDef** object.
1. Set its **Connect** and **SourceTableName** properties (and optionally, its **Attributes** property).
2. Use the **Append** method to add it to the **TableDefs** collection of a **Database**.

To refer to a **TableDef** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

TableDefs(0)
TableDefs("*name*")
TableDefs![*name*]

TableDefs Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolTableDefC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolTableDefX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolTableDefP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolTableDefM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolTableDefS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolTableDefU":1}

A **TableDefs** collection contains all stored **TableDef** objects in a database (Microsoft Jet workspaces only).

Remarks

You manipulate a table definition using a **TableDef** object and its methods and properties.

The default collection of a **Database** object is the **TableDefs** collection.

To refer to a **TableDef** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

TableDefs(0)
TableDefs("*name*")
TableDefs![*name*]

TableDef Object, TableDefs Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumTableDefC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumTableDefS"}

TableDef Object

A **TableDef** object contains these collections, methods, and properties.

Collections

Fields (Default)
Indexes
Properties

Methods

CreateField
CreateIndex
CreateProperty
OpenRecordset
RefreshLink

Properties

Attributes
ConflictTable
Connect
DateCreated
KeepLocal (user-defined)
LastUpdated
Name
RecordCount
Replicable (user-defined)
ReplicaFilter
SourceTableName
Updatable
ValidationRule
ValidationText

A **TableDef** object may also contain application-defined properties. For details on reading and setting these properties, refer to the application's online Help.

TableDefs Collection

A **TableDefs** collection is contained in each **Database** object in a Microsoft Jet database, and contains these methods and this property.

Methods

Append
Delete
Refresh

Property

Count

User Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjUserC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjUserX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjUserP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjUserM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjUserS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjUserU":1}

A **User** object represents a user account that has access permissions when a **Workspace** object operates as a secure workgroup (Microsoft Jet workspaces only).

Remarks

You use **User** objects to establish and enforce access permissions for the **Document** objects that represent databases, tables, and queries. Also, if you know the properties of a specific **User** object, you can create a new **Workspace** object that has the same access permissions as the **User** object.

You can append an existing **User** object to the **Users** collection of a **Group** object to give a user account the access permissions for that **Group** object. Alternatively, you can append the **Group** object to the **Groups** collection in a **User** object to establish membership of the user account in that group. If you use a **Users** or **Groups** collection other than the one to which you just appended an object, you may need to use the **Refresh** method.

With the properties of a **User** object, you can:

- Use the **Name** property to return the name of an existing user. You can't return the **PID** and **Password** properties of an existing **User** object.
- Use the **Name**, **PID**, and **Password** properties of a newly created, unappended **User** object to establish the identity of that **User** object. If you don't set the **Password** property, it's set to a zero-

length string ("").

The Microsoft Jet database engine predefines two **User** objects named Admin and Guest. The user Admin is a member of both of the **Group** objects named Admins and Users; the user Guest is a member only of the **Group** object named Guests.

To create a new **User** object, use the **CreateUser** method.

To refer to a **User** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

[*workspace* **|** *group*].**Users**(0)
[*workspace* **|** *group*].**Users**("*name*")
[*workspace* **|** *group*].**Users**![*name*]

Users Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolUserC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolUserX":1}
{ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolUserP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolUserM"}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolUserS"} {ewc
HLP95EN.DLL,DYNALINK,"Summary":"dacolUserU":1}

A **Users** collection contains all stored **User** objects of a **Workspace** or **Group** object (Microsoft Jet workspaces only).

Remarks

You can append an existing **User** object to the **Users** collection of a **Group** object to give a user account the access permissions for that **Group** object. Alternatively, you can append the **Group** object to the **Groups** collection in a **User** object to establish membership of the user account in that group. If you use a **Users** or **Groups** collection other than the one to which you just appended an object, you may need to use the **Refresh** method.

The Microsoft Jet database engine predefines two **User** objects named Admin and Guest. The user Admin is a member of both of the **Group** objects named Admins and Users; the user Guest is a member only of the **Group** object named Guests.

To refer to a **User** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

[*workspace* **|** *group*].**Users**(0)
[*workspace* **|** *group*].**Users**("*name*")
[*workspace* **|** *group*].**Users**![*name*]

User Object, Users Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumUserC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumUserS"}

User Object

A **User** object contains these collections, methods, and properties.

Collections

Groups (Default)

Properties

Methods

CreateGroup

NewPassword

Properties

Name

Password

PID

Users Collection

A **Users** collection is contained in each **Group** and Microsoft Jet **Workspace** object, and contains these methods and this property.

Methods

Append

Delete

Refresh

Property

Count

Workspace Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjWorkspaceC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjWorkspaceX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjWorkspaceP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjWorkspaceM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjWorkSpaceS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjWorkSpaceU":1}

A **Workspace** object defines a named session for a user. It contains open databases and provides mechanisms for simultaneous transactions and, in Microsoft Jet workspaces, secure workgroup support. It also controls whether you are going through the Microsoft Jet database engine or ODBCDirect to access external data.

Connections

Remarks

A **Workspace** is a non-persistent object that defines how your application interacts with data — either by using the Microsoft Jet database engine, or ODBCDirect. Use the **Workspace** object to manage the current session or to start an additional session. In a session, you can open multiple databases or connections, and manage transactions. For example, you can:

- Use the **Name**, **UserName**, and **Type** properties to establish a named session. The session creates a scope in which you can open multiple databases and conduct one instance of nested transactions.
- Use the **Close** method to terminate a session.
- Use the **OpenDatabase** method to open one or more existing databases on a **Workspace**.
- Use the **BeginTrans**, **CommitTrans**, and **Rollback** methods to manage nested transaction processing within a **Workspace** and use several **Workspace** objects to conduct multiple, simultaneous, and overlapping transactions.

Further, using a Microsoft Jet database, you can establish security based on user names and passwords:

- Use the **Groups** and **Users** collections to establish group and user access permissions to objects in the **Workspace**.
- Use the **IsolateODBCTrans** property to isolate multiple transactions that involve the same Microsoft Jet-connected ODBC database.

Note For a complete list of all methods, properties, and collections available on a **Workspace** object in either a Microsoft Jet database or an ODBCDirect database, see the Summary topic.

When you first refer to or use a **Workspace** object, you automatically create the default workspace, `DBEngine.Workspaces(0)`. The settings of the **Name** and **UserName** properties of the default workspace are "#Default Workspace#" and "Admin," respectively. If security is enabled, the **UserName** property setting is the name of the user who logged on.

To establish an ODBCDirect **Workspace** object, and thereby avoid loading the Microsoft Jet database engine into memory, set the **DBEngine** object's **DefaultType** property to **dbUseODBC**, or set the *type* argument of the **CreateWorkspace** method to **dbUseODBC**.

When you use transactions, all databases in the specified **Workspace** are affected — even if multiple **Database** objects are opened in the **Workspace**. For example, you use a **BeginTrans** method, update several records in a database, and then delete records in another database. If you then use the **Rollback** method, both the update and delete operations are canceled and rolled back. You can create additional **Workspace** objects to manage transactions independently across **Database** objects.

You can create **Workspace** objects with the **CreateWorkspace** method. After you create a new **Workspace** object, you must append it to the **Workspaces** collection if you need to refer to it from the **Workspaces** collection.

You can use a newly created **Workspace** object without appending it to the **Workspaces** collection. However, you must refer to it by the object variable to which you have assigned it.

To refer to a **Workspace** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

DBEngine.**Workspaces**(0)
DBEngine.**Workspaces**("*name*")
DBEngine.**Workspaces**![*name*]

Workspaces Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolWorkspaceC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolWorkspaceX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolWorkspaceP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolWorkspaceM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolWorkspaceS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolWorkspaceU":1}

A **Workspaces** collection contains all active, unhidden **Workspace** objects of the **DBEngine** object. (Hidden **Workspace** objects are not appended to the collection and referenced by the variable to which they are assigned.)

Connections

Remarks

Use the **Workspace** object to manage the current session or to start an additional session.

When you first refer to or use a **Workspace** object, you automatically create the default workspace, `DBEngine.Workspaces(0)`. The settings of the **Name** and **UserName** properties of the default workspace are "#Default Workspace#" and "Admin," respectively. If security is enabled, the **UserName** property setting is the name of the user who logged on.

You can create new **Workspace** objects with the **CreateWorkspace** method. After you create a new **Workspace** object, you must append it to the **Workspaces** collection if you need to refer to it from the **Workspaces** collection. You can, however, use a newly created **Workspace** object without appending it to the **Workspaces** collection.

To refer to a **Workspace** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

DBEngine.**Workspaces**(0)
DBEngine.**Workspaces**("*name*")
DBEngine.**Workspaces**![*name*]

Workspace Object, Workspaces Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumWorkspaceC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumWorkspaceS"}

Workspace Object

A **Workspace** object contains these collections, methods, and properties.

Legend:

Feature available in Microsoft Jet workspaces only.

Feature available in ODBCDirect workspaces only.

Collections

Connections
Databases (default)

Groups
Properties

Users

Methods

BeginTrans
Close
CommitTrans

CreateDatabase

CreateGroup

CreateUser

OpenConnection
OpenDatabase
Rollback

Properties

DefaultCursorDriver

IsolateODBCTrans

LoginTimeout
Name
Type
UserName

Workspaces Collection

A **Workspaces** collection is contained in the **DBEngine** object, and contains these methods and this

property.

Methods

Append
Delete
Refresh

Property

Count

Connection Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjConnectionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjConnectionX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjConnectionP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjConnectionM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjConnectionS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjConnectionU":1}

A **Connection** object represents a connection to an ODBC database (ODBCDirect workspaces only).

Connections

Remarks

A **Connection** is a non-persistent object that represents a connection to a remote database. The **Connection** object is only available in ODBCDirect workspaces (that is, a **Workspace** object created with the type option set to **dbUseODBC**).

Note Code written for earlier versions of DAO can continue to use the **Database** object for backward compatibility, but if the new features of a **Connection** are desired, you should revise code to use the **Connection** object. To help with code conversion, you can obtain a **Connection** object reference from a **Database** by reading the **Connection** property of the **Database** object. Conversely, you can obtain a **Database** object reference from the **Connection** object's **Database** property.

Connections Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dacolConnectionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"dacolConnectionX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"dacolConnectionP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"dacolConnectionM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dacolConnectionS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"dacolConnectionU":1}

A **Connections** collection contains the current **Connection** objects of a **Workspace** object. (ODBCDirect workspaces only).

Remarks

When you open a **Connection** object, it is automatically appended to the **Connections** collection of the **Workspace**. When you close a **Connection** object with the **Close** method, it is removed from the **Connections** collection. You should close all open **Recordset** objects within the **Connection** before closing it.

At the same time you open a **Connection** object, a corresponding **Database** object is created and appended to the **Databases** collection in the same **Workspace**, and vice versa. Similarly, when you close the **Connection**, the corresponding **Database** is deleted from the **Databases** collection, and so on.

The **Name** property setting of a **Connection** is a string that specifies the path of the database file. To refer to a **Connection** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

Connections(0)
Connections("*name*")

Connections![*name*]

Note You can open the same data source more than once, creating duplicate names in the **Connections** collection. You should assign **Connection** objects to object variables and refer to them by variable name.

Connection Object, Connections Collection Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumConnectionC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumConnectionS"}

Connection Object

The **Connection** object contains these collections, methods, and properties.

Collections

QueryDefs (default)

Recordsets

Methods

Cancel

Close

CreateQueryDef

Execute

OpenRecordset

Properties

Connect

Database

Name

QueryTimeout

RecordsAffected

StillExecuting

Transactions

Updatable

Connections Collection

A **Connections** collection is contained in each ODBCDirect **Workspace** object, and contains this method and this property:

Method

Refresh

Property

Count

Forward-Only–Type Recordset Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjForwardOnlyTypeRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjForwardOnlyTypeRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjForwardOnlyTypeRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjForwardOnlyTypeRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjForwardOnlyTypeRecordsetS"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjForwardOnlyTypeRecordsetU":1}

This **Recordset** type is identical to a snapshot except that you can only scroll forward through its records. This improves performance in situations where you only need to make a single pass through a result set.

In an ODBCDirect workspace, this type corresponds to an ODBC forward-only cursor.

Forward-Only–Type Recordset Object Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumForwardOnlyTypeRecordsetC "} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumForwardOnlyTypeRecordsetS"}

The forward-only type **Recordset** object contains these collections, methods, and properties.

Legend:

Feature available in Microsoft Jet workspaces only.

Feature available in ODBCDirect workspaces only.

Collections
Fields (default)
Properties

Methods	Restrictions
AddNew	
Cancel	
CancelUpdate	
Close	
CopyQueryDef	
Delete	
Edit	
GetRows	
Move	Only with forward moves that don't use a bookmark offset.
MoveNext	
NextRecordset	
Requery	
Update	

Properties

The following table indicates whether each property setting is read/write, read-only, or always **False** in either Microsoft Jet or ODBCDirect workspaces.

Read-only

Read/write

Properties	Restrictions
BatchCollisionCount	
BatchCollisions	
BatchSize	
BOF	

Property	Value
Connection	
EOF	
Filter	
Name	
RecordCount	
RecordStatus	
Restartable	
StillExecuting	
Transactions	Always **False**
Type	
Updatable	**False**
UpdateOptions	
ValidationRule	
ValidationText	

Dynamic-Type Recordset Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daobjDynamicTypeRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daobjDynamicTypeRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"daobjDynamicTypeRecordsetP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"daobjDynamicTypeRecordsetM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daobjDynamicTypeRecordset S"} {ewc HLP95EN.DLL,DYNALINK,"Summary":"daobjDynamicTypeRecordsetU":1}

This **Recordset** type represents a query result set from one or more base tables in which you can add, change, or delete records from a row-returning query. Further, records that other users add, delete, or edit in the base tables also appear in your **Recordset**.

This type is only available in ODBCDirect workspaces, and corresponds to an ODBC dynamic cursor.

Dynamic-Type Recordset Object Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumDynamicTypeRecordsetC "} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumDynamicTypeRecordsetS"}

A dynamic-type **Recordset** object contains these collections, methods, and properties. This type of **Recordset** and its methods and properties are available only in an ODBCDirect workspace.

Collections

Fields (default)

Properties

Methods

AddNew

Cancel

CancelUpdate

Close

Delete

Edit

GetRows

Move

MoveFirst

MoveLast

MoveNext

MovePrevious

NextRecordset

Requery

Update

Properties

The following table indicates whether each property setting is read/write, read-only, or always **False**.

Read-only

Read/write

Properties	Restrictions
AbsolutePosition	
BatchCollisionCount	
BatchCollisions	
BatchSize	
BOF	
Bookmark	
Bookmarkable	
CacheSize	
Connection	

EditMode

EOF

LastModified

LockEdits

Name

PercentPosition

RecordCount

RecordStatus

Restartable

StillExecuting

Type

Updatable

UpdateOptions

Dynaset-Type Recordset Object Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumDynasetTypeRecordsetC "} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumDynasetTypeRecordsetS"}

The dynaset-type **Recordset** object contains these collections, methods, and properties.

Legend:

Feature available in Microsoft Jet workspaces only.

Feature available in ODBCDirect workspaces only.

Collections

Fields (default)
Properties

Methods	**Restrictions**
AddNew	
Cancel	
CancelUpdate	
Clone	
Close	
CopyQueryDef	
Delete	
Edit	
FillCache	
FindFirst	
FindLast	
FindNext	
FindPrevious	
GetRows	
Move	
MoveFirst	
MoveLast	
MoveNext	
MovePrevious	
NextRecordset	
OpenRecordset	
Requery	
Update	

Properties

The following table indicates whether the property setting is read/write, read-only, or only available in either Microsoft Jet or ODBCDirect workspaces.

Read-only

Read/write

Properties	Restrictions
AbsolutePosition	
BatchCollisionCount	
BatchCollisions	
BatchSize	
BOF	
Bookmark	
Bookmarkable	
CacheSize	Microsoft Jet ODBCDirect
CacheStart	
Connection	
EditMode	
EOF	
Filter	
LastModified	
LockEdits	
Name	
NoMatch	
PercentPosition	
RecordCount	
RecordStatus	
Restartable	
Sort	
StillExecuting	
Transactions	
Type	
Updatable	
UpdateOptions	
ValidationRule	

ValidationText

Snapshot-Type Recordset Object Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumSnapshotTypeRecordsetC "} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumSnapshotTypeRecordsetS"}

The snapshot-type **Recordset** object contains these collections, methods, and properties.

Legend:

Feature available in Microsoft Jet workspaces only.

Feature available in ODBCDirect workspaces only.

Collections
Fields (default)
Properties

Methods	Restrictions
AddNew	*
Cancel	
CancelUpdate	*
Clone	
Close	
CopyQueryDef	
Delete	*
Edit	*
FindFirst	
FindLast	
FindNext	
FindPrevious	
GetRows	
Move	
MoveFirst	
MoveLast	
MoveNext	
MovePrevious	
NextRecordset	
OpenRecordset	
Requery	
Update	*

* In an ODBCDirect workspace, a snapshot-type **Recordset** may be updatable, depending on the ODBC driver. The **AddNew**, **Edit**, **Delete**, **Update**, and **CancelUpdate** methods are only available on ODBCDirect snapshot-type **Recordset** objects if the ODBC driver supports updatable snapshots.

Properties

The following table indicates whether the property setting is read/write, read-only, or always **False** in either Microsoft Jet or ODBCDirect workspaces.

Properties	Read-only	Read/write	Restrictions
AbsolutePosition			
BatchCollisionCount			
BatchCollisions			
BatchSize			
BOF			
Bookmark			
Bookmarkable			
CacheSize			
Connection			
EditMode			
EOF			
Filter			
LastModified			
LockEdits			*
Name			
NoMatch			
PercentPosition			
RecordCount			
RecordStatus			
Restartable			
Sort			
StillExecuting			
Transactions			Always **False**
Type			
Updatable			Always **False** in Microsoft Jet workspaces; in ODBCDirect workspaces *

UpdateOptions

ValidationRule

ValidationText

---

* In an ODBCDirect workspace, a snapshot-type **Recordset** may be updatable, depending on the ODBC driver. The **LastModified** property is available, and the **Updatable** property is **True** only on ODBCDirect snapshot-type **Recordset** objects if the ODBC driver supports updatable snapshots.

Table-Type Recordset Object Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"dasumTableTypeRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"dasumTableTypeRecordsetS"}

A table-type **Recordset** object contains these collections, methods, and properties. This type of **Recordset** and its methods and properties are available only in a Microsoft Jet workspace.

Collections

Fields (default)

Properties

Methods

AddNew

CancelUpdate

Clone

Close

Delete

Edit

GetRows

Move

MoveFirst

MoveLast

MoveNext

MovePrevious

OpenRecordset

Seek

Update

Properties

The following table indicates whether each property setting is read/write, read-only, or always **False**.

Read-only

Read/write

Properties	Restrictions
BOF	
Bookmark	
Bookmarkable	
DateCreated	
EditMode	
EOF	
Index	
LastModified	
LastUpdated	

LockEdits	
Name	
NoMatch	
PercentPosition	
RecordCount	
Restartable	Always **False**
Transactions	
Type	
Updatable	
ValidationRule	
ValidationText	

AddNew Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthAddNewC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthAddNewX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthAddNewA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthAddNewS"}

Creates a new record for an updatable **Recordset** object.

Syntax

*recordset***.AddNew**

The *recordset* placeholder is an object variable that represents an updatable **Recordset** object to which you want to add a new record.

Remarks

Use the **AddNew** method to create and add a new record in the **Recordset** object named by *recordset*. This method sets the fields to default values, and if no default values are specified, it sets the fields to **Null** (the default values specified for a table-type **Recordset**).

After you modify the new record, use the **Update** method to save the changes and add the record to the **Recordset**. No changes occur in the database until you use the **Update** method.

Caution If you issue an **AddNew** and then perform any operation that moves to another record, but without using **Update**, your changes are lost without warning. In addition, if you close the **Recordset** or end the procedure that declares the **Recordset** or its **Database** object, the new record is discarded without warning.

Note When you use **AddNew** in a Microsoft Jet workspace and the database engine has to create a new page to hold the current record, page locking is pessimistic. If the new record fits in an existing page, page locking is optimistic.

If you haven't moved to the last record of your **Recordset**, records added to base tables by other processes may be included if they are positioned beyond the current record. If you add a record to your own **Recordset**, however, the record is visible in the **Recordset** and included in the underlying table where it becomes visible to any new **Recordset** objects.

The position of the new record depends on the type of **Recordset**:

- In a dynaset-type **Recordset** object, records are inserted at the end of the **Recordset**, regardless of any sorting or ordering rules that were in effect when the **Recordset** was opened.
- In a table-type **Recordset** object whose **Index** property has been set, records are returned in their proper place in the sort order. If you haven't set the **Index** property, new records are returned at the end of the **Recordset**.

The record that was current before you used **AddNew** remains current. If you want to make the new record current, you can set the **Bookmark** property to the bookmark identified by the **LastModified** property setting.

Note To add, edit, or delete a record, there must be a unique index on the record in the underlying data source. If not, a "Permission denied" error will occur on the **AddNew**, **Delete**, or **Edit** method call in a Microsoft Jet workspace, or an "Invalid argument" error will occur on the **Update** call in an ODBCDirect workspace.

Append Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthAppendC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthAppendX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthAppendA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthAppendS"}

Adds a new DAO object to a collection.

Syntax

collection.**Append** *object*

The **Append** method syntax has these parts.

Part	Description
collection	An object variable that represents any collection that can accept new objects (for limitations, see the table at the end of this topic).
object	An object variable that represents the object being appended, which must be of the same type as the elements of *collection.*

Remarks

You can use the **Append** method to add a new table to a database, add a field to a table, and add a field to an index.

The appended object becomes a persistent object, stored on disk, until you delete it by using the **Delete** method. If *collection* is a **Workspaces** collection (which is stored only in memory), the object is active until you remove it by using the **Close** method.

The addition of a new object occurs immediately, but you should use the **Refresh** method on any other collections that may be affected by changes to the database structure.

If the object you're appending isn't complete (such as when you haven't appended any **Field** objects to a **Fields** collection of an **Index** object before it's appended to an **Indexes** collection) or if the properties set in one or more subordinate objects are incorrect, using the **Append** method causes an error. For example, if you haven't specified a field type and then try to append the **Field** object to the **Fields** collection in a **TableDef** object, using the **Append** method triggers a run-time error.

The following table lists some limitations of the **Append** method. The object in the first column is an object containing the collection in the second column. The third column indicates whether you can append an object to that collection (for example, you can never append a **Container** object to the **Containers** collection of a **Database** object).

Object	Collection	Can you append new objects?
DBEngine	**Workspaces**	Yes
DBEngine	**Errors**	No. New Error objects are automatically appended when they occur.
Workspace	**Connections**	No. Using the **OpenConnection** method automatically appends new objects.
Workspace	**Databases**	No. Using the **OpenDatabase** method automatically appends new

		objects.
Workspace	**Groups**	Yes
Workspace	**Users**	Yes
Connection	**QueryDefs**	No. Using the **CreateQueryDef** method automatically appends new objects.
Connection	**Recordsets**	No. Using the **OpenRecordset** method automatically appends new objects.
Database	**Containers**	No
Database	**QueryDefs**	Only when the **QueryDef** object is a new, unappended object created with no name. See the **CreateQueryDef** method for details.
Database	**Recordsets**	No. Using the **OpenRecordset** method automatically appends new objects.
Database	**Relations**	Yes
Database	**TableDefs**	Yes
Group	**Users**	Yes
User	**Groups**	Yes
Container	**Documents**	No
QueryDef	**Fields**	No
QueryDef	**Parameters**	No
Recordset	**Fields**	No
Relation	**Fields**	Yes
TableDef	**Fields**	Only when the **Updatable** property of the **TableDef** object is set to **True**, or when the **TableDef** object is unappended.
TableDef	**Indexes**	Only when the **Updatable** property of the **TableDef** is set to **True**, or when the **TableDef** object is unappended.
Index	**Fields**	Only when the **Index** object is a new, unappended object.
Database, **Field**, **Index**, **QueryDef**, **TableDef**	**Properties**	Only when the **Database**, **Field**, **Index**, **QueryDef**, or **TableDef** object is in a Microsoft Jet workspace.
DBEngine, **Parameter**, **Recordset**,	**Properties**	No

Workspace

AppendChunk Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthAppendChunkC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthAppendChunkX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthAppendChunkA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthAppendChunkS"}

Appends data from a string expression to a Memo or Long Binary **Field** object in a **Recordset**.

Syntax

recordset **!** *field***.AppendChunk** *source*

The **AppendChunk** method syntax has these parts.

Part	Description
recordset	An object variable that represents the **Recordset** object containing the **Fields** collection.
field	An object variable that represents the name of a **Field** object whose **Type** property is set to **dbMemo** (Memo), **dbLongBinary** (Long Binary), or the equivalent.
source	A **Variant** (**String** subtype) expression or variable containing the data you want to append to the **Field** object specified by *field.*

Remarks

You can use the **AppendChunk** and **GetChunk** methods to access subsets of data in a Memo or Long Binary field.

You can also use these methods to conserve string space when you work with Memo and Long Binary fields. Certain operations (copying, for example) involve temporary strings. If string space is limited, you may need to work with chunks of a field instead of the entire field.

If there is no current record when you use **AppendChunk**, an error occurs.

Notes

- The initial **AppendChunk** operation (after an **Edit** or **AddNew** call) will simply place the data in the field, overwriting any existing data. Subsequent **AppendChunk** calls within the same **Edit** or **AddNew** session will then add to the existing data.
- In an ODBCDirect workspace, unless you first edit another field in the current record, using **AppendChunk** will fail (though no error occurs) while you are in **Edit** mode.
- In an ODBCDirect workspace, after you use **AppendChunk** on a field, you cannot read or write that field in an assignment statement until you move off the current record and then return to it. You can do this by using the **MoveNext** and **MovePrevious** methods.

BeginTrans, CommitTrans, Rollback Methods

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthBeginTransC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthBeginTransX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthBeginTransA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthBeginTransS"}

The transaction methods manage transaction processing during a session defined by a **Workspace** object as follows:

- **BeginTrans** begins a new transaction.
- **CommitTrans** ends the current transaction and saves the changes.
- **Rollback** ends the current transaction and restores the databases in the **Workspace** object to the state they were in when the current transaction began.

Syntax

workspace.**BeginTrans | CommitTrans** [**dbFlushOSCacheWrites**] **| Rollback**

The *workspace* placeholder is an object variable that represents the **Workspace** containing the databases that will use transactions.

Remarks

You use these methods with a **Workspace** object when you want to treat a series of changes made to the databases in a session as one unit.

Typically, you use transactions to maintain the integrity of your data when you must both update records in two or more tables and ensure changes are completed (committed) in all tables or none at all (rolled back). For example, if you transfer money from one account to another, you might subtract an amount from one and add the amount to another. If either update fails, the accounts no longer balance. Use the **BeginTrans** method before updating the first record, and then, if any subsequent update fails, you can use the **Rollback** method to undo all of the updates. Use the **CommitTrans** method after you successfully update the last record.

In a Microsoft Jet workspace, you can include the **dbFlushOSCacheWrites** constant with **CommitTrans**, This forces the database engine to immediately flush all updates to disk, instead of caching them temporarily. Without using this option, a user could get control back immediately after the application program calls **CommitTrans**, turn the computer off, and not have the data written to disk. While using this option may affect your application's performance, it is useful in situations where the computer could be shut off before cached updates are saved to disk.

Caution Within one **Workspace** object, transactions are always global to the **Workspace** and aren't limited to only one **Connection** or **Database** object. If you perform operations on more than one connection or database within a **Workspace** transaction, resolving the transaction (that is, using the **CommitTrans** or **Rollback** method) affects all operations on all connections and databases within that workspace.

After you use **CommitTrans**, you can't undo changes made during that transaction unless the transaction is nested within another transaction that is itself rolled back. If you nest transactions, you must resolve the current transaction before you can resolve a transaction at a higher level of nesting.

If you want to have simultaneous transactions with overlapping, non-nested scopes, you can create additional **Workspace** objects to contain the concurrent transactions.

If you close a **Workspace** object without resolving any pending transactions, the transactions are automatically rolled back.

If you use the **CommitTrans** or **Rollback** method without first using the **BeginTrans** method, an error occurs.

Some ISAM databases used in a Microsoft Jet workspace may not support transactions, in which

case the **Transactions** property of the **Database** object or **Recordset** object is **False**. To make sure the database supports transactions, check the value of the **Transactions** property of the **Database** object before using the **BeginTrans** method. If you are using a **Recordset** object based on more than one database, check the **Transactions** property of the **Recordset** object. If a **Recordset** is based entirely on Microsoft Jet tables, you can always use transactions. **Recordset** objects based on tables created by other database products, however, may not support transactions. For example, you can't use transactions in a **Recordset** based on a Paradox table. In this case, the **Transactions** property is **False**. If the **Database** or **Recordset** doesn't support transactions, the methods are ignored and no error occurs.

You can't nest transactions if you are accessing ODBC data sources through the Microsoft Jet database engine.

Notes

- You can often improve the performance of your application by breaking operations that require disk access into transaction blocks. This buffers your operations and may significantly reduce the number of times a disk is accessed.
- In a Microsoft Jet workspace, transactions are logged in a file kept in the directory specified by the TEMP environment variable on the workstation. If the transaction log file exhausts the available storage on your TEMP drive, the database engine triggers a run-time error. At this point, if you use **CommitTrans**, an indeterminate number of operations are committed, but the remaining uncompleted operations are lost, and the operation has to be restarted. Using a **Rollback** method releases the transaction log and rolls back all operations in the transaction.

CancelUpdate Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCancelUpdateC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCancelUpdateX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCancelUpdateA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCancelUpdateS"}

Cancels any pending updates for a **Recordset** object.

Syntax

*recordset***.CancelUpdate** *type*

The **AppendChunk** method syntax has these parts.

Part	Description
recordset	An object variable that represents the **Recordset** object for which you are canceling pending updates.
type	Optional. A constant indicating the type of update, as specified in Settings (ODBCDirect workspaces only).

Settings

You can use the following values for the *type* argument only if batch updating is enabled.

Constant	Description
dbUpdateRegular	Default. Cancels pending changes that aren’t cached.
dbUpdateBatch	Cancels pending changes in the update cache.

Remarks

You can use the **CancelUpdate** method to cancel any pending updates resulting from an **Edit** or **AddNew** operation. For example, if a user invokes the **Edit** or **AddNew** method and hasn't yet invoked the **Update** method, **CancelUpdate** cancels any changes made after **Edit** or **AddNew** was invoked.

Check the **EditMode** property of the **Recordset** to determine if there is a pending operation that can be canceled.

Note Using the **CancelUpdate** method has the same effect as moving to another record without using the **Update** method, except that the current record doesn't change, and various properties, such as **BOF** and **EOF**, aren't updated.

Clone Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCloneC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCloneX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCloneA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCloneS"}

Creates a duplicate **Recordset** object that refers to the original **Recordset** object.

Syntax

Set *duplicate* = *original*.**Clone**

The **Clone** method syntax has these parts.

Part	Description
duplicate	An object variable identifying the duplicate **Recordset** object you're creating.
original	An object variable identifying the **Recordset** object you want to duplicate.

Remarks

Use the **Clone** method to create multiple, duplicate **Recordset** objects. Each **Recordset** can have its own current record. Using **Clone** by itself doesn't change the data in the objects or in their underlying structures. When you use the **Clone** method, you can share bookmarks between two or more **Recordset** objects because their bookmarks are interchangeable.

You can use the **Clone** method when you want to perform an operation on a **Recordset** that requires multiple current records. This is faster and more efficient than opening a second **Recordset**. When you create a **Recordset** with the **Clone** method, it initially lacks a current record. To make a record current before you use the **Recordset** clone, you must set the **Bookmark** property or use one of the Move methods, one of the Find methods, or the **Seek** method.

Using the **Close** method on either the original or duplicate object doesn't affect the other object. For example, using **Close** on the original **Recordset** doesn't close the clone.

Notes

- Closing a clone **Recordset** within a pending transaction will cause an implicit **Rollback** operation.
- When you clone a table-type **Recordset** object in a Microsoft Jet workspace, the **Index** property setting is not cloned on the new copy of the **Recordset**. You must copy the **Index** property setting manually.
- You can use the **Clone** method with forward-only–type **Recordset** objects only in an ODBCDirect workspace.

Close Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCloseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCloseX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCloseA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCloseS"}

Closes an open DAO object.

Syntax

*object***.Close**

The *object* placeholder is an object variable that represents an open **Connection**, **Database**, **Recordset**, or **Workspace** object.

Remarks

Closing an open object removes it from the collection to which it's appended. Any attempt to close the default workspace is ignored.

If the **Connection**, **Database**, **Recordset**, or **Workspace** object named by *object* is already closed when you use **Close**, a run-time error occurs.

Caution If you exit a procedure that declares **Connection**, **Database**, or **Recordset** objects, those objects are closed, all pending transactions are rolled back, and any pending edits to your data are lost.

If you try to close a **Connection** or **Database** object while it has any open **Recordset** objects, the **Recordset** objects will be closed and any pending updates or edits will be canceled. Similarly, if you try to close a **Workspace** object while it has any open **Connection** or **Database** objects, those **Connection** and **Database** objects will be closed, which will close their **Recordset** objects.

Using the **Close** method on either an original or cloned **Recordset** object doesn't affect the other **Recordset** object.

To remove objects from updatable collections other than the **Connections**, **Databases**, **Recordsets**, and **Workspaces** collections, use the **Delete** method on those collections. You can't add a new member to the **Containers**, **Documents**, and **Errors** collections.

An alternative to the **Close** method is to set the value of an object variable to **Nothing** (`Set dbsTemp = Nothing`).

CompactDatabase Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCompactDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCompactDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCompactDatabaseA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCompactDatabaseS"}

Copies and compacts a closed database, and gives you the option of changing its version, collating order, and encryption. (Microsoft Jet workspaces only).

Syntax

DBEngine.CompactDatabase *olddb*, *newdb*, *locale*, *options*, *password*

The **CompactDatabase** method syntax has these parts.

Part	Description
olddb	A **String** that identifies an existing, closed database. It can be a full path and file name, such as `"C:\db1.mdb"`. If the file name has an extension, you must specify it. If your network supports it, you can also specify a network path, such as `"\\server1\share1\dir1\db1.mdb"`.
newdb	A **String** that is the file name (and path) of the compacted database that you're creating. You can also specify a network path. You can't use the *newdb* argument to specify the same database file as *olddb*.

locale	Optional. A **Variant** that is a string expression that specifies a collating order for creating *newdb*, as specified in Settings. If you omit this argument, the locale of *newdb* is the same as *olddb*.
	You can also create a password for *newdb* by concatenating the password string (starting with `";pwd="`) with a constant in the *locale* argument, like this:
	`dbLangSpanish & ";pwd=NewPassword"`
	If you want to use the same *locale* as *olddb* (the default value), but specify a new password, simply enter a password string for *locale*:
	`";pwd=NewPassword"`

options	Optional. A constant or combination of constants that indicates one or more options, as specified in Settings. You can combine options by summing the corresponding constants.
password	Optional. A **Variant** that is a string expression containing a password, if the database is password protected. The string `";pwd="` must precede the actual password. If you include a password setting in *locale*, this setting is ignored.

Settings

You can use one of the following constants for the *locale* argument to specify the **CollatingOrder** property for string comparisons of text.

Constant	Collating order
dbLangGeneral	English, German, French, Portuguese,

	Italian, and Modern Spanish
dbLangArabic	Arabic
dbLangChineseSimplified	Simplified Chinese
dbLangChineseTraditional	Traditional Chinese
dbLangCyrillic	Russian
dbLangCzech	Czech
dbLangDutch	Dutch
dbLangGreek	Greek
dbLangHebrew	Hebrew
dbLangHungarian	Hungarian
dbLangIcelandic	Icelandic
dbLangJapanese	Japanese
dbLangKorean	Korean
dbLangNordic	Nordic languages (Microsoft Jet database engine version 1.0 only)
dbLangNorwDan	Norwegian and Danish
dbLangPolish	Polish
dbLangSlovenian	Slovenian
dbLangSpanish	Traditional Spanish
dbLangSwedFin	Swedish and Finnish
dbLangThai	Thai
dbLangTurkish	Turkish

You can use one of the following constants in the *options* argument to specify whether to encrypt or to decrypt the database while it's compacted.

Constant	Description
dbEncrypt	Encrypt the database while compacting.
dbDecrypt	Decrypt the database while compacting.

If you omit an encryption constant or if you include both **dbDecrypt** and **dbEncrypt**, *newdb* will have the same encryption as *olddb.*

You can use one of the following constants in the *options* argument to specify the version of the data format for the compacted database. This constant affects only the version of the data format of *newdb* and doesn't affect the version of any Microsoft Access-defined objects, such as forms and reports.

Constant	Description
dbVersion10	Creates a database that uses the Microsoft Jet database engine version 1.0 file format while compacting.
dbVersion11	Creates a database that uses the Microsoft Jet database engine version 1.1 file format while compacting.
dbVersion20	Creates a database that uses the Microsoft Jet database engine version 2.0 file format while compacting.
dbVersion30	Creates a database that uses the Microsoft Jet database engine version 3.0 file format (compatible with version 3.5) while compacting.

You can specify only one version constant. If you omit a version constant, *newdb* will have the same

version as *olddb*. You can compact *newdb* only to a version that is the same or later than that of *olddb*.

Remarks

As you change data in a database, the database file can become fragmented and use more disk space than is necessary. Periodically, you can use the **CompactDatabase** method to compact your database to defragment the database file. The compacted database is usually smaller and often runs faster. You can also change the collating order, the encryption, or the version of the data format while you copy and compact the database.

You must close *olddb* before you compact it. In a multiuser environment, other users can't have *olddb* open while you're compacting it. If *olddb* isn't closed or isn't available for exclusive use, an error occurs.

Because **CompactDatabase** creates a copy of the database, you must have enough disk space for both the original and the duplicate databases. The compact operation fails if there isn't enough disk space available. The *newdb* duplicate database doesn't have to be on the same disk as *olddb*. After successfully compacting a database, you can delete the *olddb* file and rename the compacted *newdb* file to the original file name.

The **CompactDatabase** method copies all the data and the security permission settings from the database specified by *olddb* to the database specified by *newdb*.

If you use **CompactDatabase** to convert a version 1.*x* database to version 2.5 or 3.*x*, only applications using version Microsoft Jet 2.5 or 3.*x* can open the converted database.

Note In an ODBCDirect workspace, using the **CompactDatabase** method doesn't return an error, but instead loads the Microsoft Jet database engine into memory.

Caution Because the **CompactDatabase** method doesn't convert Microsoft Access objects, you shouldn't use **CompactDatabase** to convert a database containing such objects. To convert a database containing Microsoft Access objects, on the **Tools** menu, point to **Database Utilities**, and then click **Convert Database**.

CopyQueryDef Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCopyQueryDefC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCopyQueryDefX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCopyQueryDefA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCopyQueryDefS"}

Returns a **QueryDef** object that is a copy of the **QueryDef** used to create the **Recordset** object represented by the *recordset* placeholder (Microsoft Jet workspaces only).

Syntax

Set *querydef* = *recordset***.CopyQueryDef**

The **CopyQueryDef** method syntax has these parts.

Part	Description
querydef	An object variable that represents the copy of a **QueryDef** object you want to create.
recordset	An object variable that represents the **Recordset** object created with the original **QueryDef** object.

Remarks

You can use the **CopyQueryDef** method to create a new **QueryDef** that is a duplicate of the **QueryDef** used to create the **Recordset**.

If a **QueryDef** wasn't used to create this **Recordset**, an error occurs. You must first open a **Recordset** with the **OpenRecordset** method before using the **CopyQueryDef** method.

This method is useful when you create a **Recordset** object from a **QueryDef**, and pass the **Recordset** to a function, and the function must re-create the SQL equivalent of the query, for example, to modify it in some way.

CreateDatabase Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCreateDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCreateDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCreateDatabaseA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCreateDatabaseS"}

Creates a new **Database** object, saves the database to disk, and returns an opened **Database** object (Microsoft Jet workspaces only).

Syntax

Set *database* = *workspace*.**CreateDatabase (***name*, *locale*, *options***)**

The **CreateDatabase** method syntax has these parts.

Part	Description
database	An object variable that represents the **Database** object you want to create.
workspace	An object variable that represents the existing **Workspace** object that will contain the database. If you omit *workspace*, **CreateDatabase** uses the default **Workspace**.
name	A **String** up to 255 characters long that is the name of the database file that you're creating. It can be the full path and file name, such as `"C:\db1.mdb"`. If you don't supply a file name extension, .mdb is appended. If your network supports it, you can also specify a network path, such as `"\\server1\share1\dir1\db1"`. You can only create .mdb database files with this method.

locale	A string expression that specifies a collating order for creating the database, as specified in Settings. You must supply this argument or an error occurs.
	You can also create a password for the new **Database** object by concatenating the password string (starting with `";pwd="`) with a constant in the *locale* argument, like this:
	`dbLangSpanish & ";pwd=NewPassword"`
	If you want to use the default *locale*, but specify a password, simply enter a password string for the *locale* argument:
	`";pwd=NewPassword"`

options	Optional. A constant or combination of constants that indicates one or more options, as specified in Settings. You can combine options by summing the corresponding constants.

Settings

You can use one of the following constants for the *locale* argument to specify the **CollatingOrder** property of text for string comparisons.

Constant	Collating order
dbLangGeneral	English, German, French, Portuguese, Italian, and Modern Spanish
dbLangArabic	Arabic

dbLangChineseSimplified	Simplified Chinese
dbLangChineseTraditional	Traditional Chinese
dbLangCyrillic	Russian
dbLangCzech	Czech
dbLangDutch	Dutch
dbLangGreek	Greek
dbLangHebrew	Hebrew
dbLangHungarian	Hungarian
dbLangIcelandic	Icelandic
dbLangJapanese	Japanese
dbLangKorean	Korean
dbLangNordic	Nordic languages (Microsoft Jet database engine version 1.0 only)
dbLangNorwDan	Norwegian and Danish
dbLangPolish	Polish
dbLangSlovenian	Slovenian
dbLangSpanish	Traditional Spanish
dbLangSwedFin	Swedish and Finnish
dbLangThai	Thai
dbLangTurkish	Turkish

You can use one or more of the following constants in the *options* argument to specify which version the data format should have and whether or not to encrypt the database.

Constant	Description
dbEncrypt	Creates an encrypted database.
dbVersion10	Creates a database that uses the Microsoft Jet database engine version 1.0 file format.
dbVersion11	Creates a database that uses the Microsoft Jet database engine version 1.1 file format.
dbVersion20	Creates a database that uses the Microsoft Jet database engine version 2.0 file format.
dbVersion30	(Default) Creates a database that uses the Microsoft Jet database engine version 3.0 file format (compatible with version 3.5).

If you omit the encryption constant, **CreateDatabase** creates an un-encrypted database. You can specify only one version constant. If you omit a version constant, **CreateDatabase** creates a database that uses the Microsoft Jet database engine version 3.0 file format.

Remarks

Use the **CreateDatabase** method to create and open a new, empty database, and return the **Database** object. You must complete its structure and content by using additional DAO objects. If you want to make a partial or complete copy of an existing database, you can use the **CompactDatabase** method to make a copy that you can customize.

CreateField Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCreateFieldC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCreateFieldX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCreateFieldA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCreateFieldS"}

Creates a new **Field** object (Microsoft Jet workspaces only).

Syntax

Set *field* = *object***.CreateField (***name*, *type*, *size***)**

The **CreateField** method syntax has these parts.

Part	Description
field	An object variable that represents the **Field** object you want to create.
object	An object variable that represents the **Index**, **Relation**, or **TableDef** object for which you want to create the new **Field** object.
name	Optional. A **Variant** (**String** subtype) that uniquely names the new **Field** object. See the **Name** property for details on valid **Field** names.
type	Optional. A constant that determines the data type of the new **Field** object. See the **Type** property for valid data types.
size	Optional. A **Variant** (**Integer** subtype) that indicates the maximum size, in bytes, of a **Field** object that contains text. See the **Size** property for valid *size* values. This argument is ignored for numeric and fixed-width fields.

Remarks

You can use the **CreateField** method to create a new field, as well as specify the name, data type, and size of the field. If you omit one or more of the optional parts when you use **CreateField**, you can use an appropriate assignment statement to set or reset the corresponding property before you append the new object to a collection. After you append the new object, you can alter some but not all of its property settings. See the individual property topics for more details.

The *type* and *size* arguments apply only to **Field** objects in a **TableDef** object. These arguments are ignored when a **Field** object is associated with an **Index** or **Relation** object.

If *name* refers to an object that is already a member of the collection, a run-time error occurs when you use the **Append** method.

To remove a **Field** object from a **Fields** collection, use the **Delete** method on the collection. You can't delete a **Field** object from a **TableDef** object's **Fields** collection after you create an index that references the field.

CreateGroup Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCreateGroupC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCreateGroupX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCreateGroupA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCreateGroupS"}

Creates a new **Group** object (Microsoft Jet workspaces only).

Syntax

Set *group* = *object***.CreateGroup (***name*, *pid***)**

The **CreateGroup** method syntax has these parts.

Part	Description
group	An object variable that represents the **Group** you want to create.
object	An object variable that represents the **User** or **Workspace** object for which you want to create the new **Group** object.
name	Optional. A **Variant** (**String** subtype) that uniquely names the new **Group** object. See the **Name** property for details on valid **Group** names.
pid	Optional. A **Variant** (**String** subtype) containing the PID of a group account. The identifier must contain from 4 to 20 alphanumeric characters. See the **PID** property for more information on valid personal identifiers.

Remarks

You can use the **CreateGroup** method to create a new **Group** object for a **User** or **Workspace**. If you omit one or both of the optional parts when you use **CreateGroup**, you can use an appropriate assignment statement to set or reset the corresponding property before you append the new object to a collection. After you append the object, you can alter some but not all of its property settings. See the individual property topics for more details.

If *name* refers to an object that is already a member of the collection, a run-time error occurs when you use the **Append** method.

To remove a **Group** object from a collection, use the **Delete** method on the **Groups** collection.

CreateIndex Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCreateIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCreateIndexX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCreateIndexA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCreateIndexS"}

Creates a new **Index** object (Microsoft Jet workspaces only).

Syntax

Set *index* = *tabledef***.CreateIndex (***name***)**

The **CreateIndex** method syntax has these parts.

Part	Description
index	An object variable that represents the index you want to create.
tabledef	An object variable that represents the **TableDef** object you want to use to create the new **Index** object.
name	Optional. A **Variant** (**String** subtype) that uniquely names the new **Index** object. See the **Name** property for details on valid **Index** names.

Remarks

You can use the **CreateIndex** method to create a new **Index** object for a **TableDef** object. If you omit the optional *name* part when you use **CreateIndex**, you can use an appropriate assignment statement to set or reset the **Name** property before you append the new object to a collection. After you append the object, you may or may not be able to set its **Name** property, depending on the type of object that contains the **Indexes** collection. See the **Name** property topic for more details.

If *name* refers to an object that is already a member of the collection, a run-time error occurs when you use the **Append** method.

To remove an **Index** object from a collection, use the **Delete** method on the collection.

CreateProperty Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCreatePropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCreatePropertyX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCreatePropertyA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCreatePropertyS"}

Creates a new user-defined **Property** object (Microsoft Jet workspaces only).

Syntax

Set *property* = *object*.**CreateProperty (***name*, *type*, *value*, *DDL***)**

The **CreateProperty** method syntax has these parts.

Part	Description
property	An object variable that represents the **Property** object you want to create.
object	An object variable that represents the **Database**, **Field**, **Index**, **QueryDef**, **Document**, or **TableDef** object you want to use to create the new **Property** object.
name	Optional. A **Variant** (**String** subtype) that uniquely names the new **Property** object. See the **Name** property for details on valid **Property** names.
type	Optional. A constant that defines the data type of the new **Property** object. See the **Type** property for valid data types.
value	Optional. A **Variant** containing the initial property value. See the **Value** property for details.
DDL	Optional. A **Variant** (**Boolean** subtype) that indicates whether or not the **Property** is a DDL object. The default is **False**. If *DDL* is **True**, users can't change or delete this **Property** object unless they have **dbSecWriteDef** permission.

Remarks

You can create a user-defined **Property** object only in the **Properties** collection of an object that is persistent.

If you omit one or more of the optional parts when you use **CreateProperty**, you can use an appropriate assignment statement to set or reset the corresponding property before you append the new object to a collection. After you append the object, you can alter some but not all of its property settings. See the **Name**, **Type**, and **Value** property topics for more details.

If *name* refers to an object that is already a member of the collection, a run-time error occurs when you use the **Append** method.

To remove a user-defined **Property** object from the collection, use the **Delete** method on the **Properties** collection. You can't delete built-in properties.

Note If you omit the *DDL* argument, it defaults to **False** (non-DDL). Because no corresponding DDL property is exposed, you must delete and re-create a **Property** object you want to change from DDL to non-DDL.

CreateQueryDef Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCreateQueryDefC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCreateQueryDefX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCreateQueryDefA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCreateQueryDefS"}

Creates a new **QueryDef** object in a specified **Connection** or **Database** object.

Syntax

Set *querydef* = *object***.CreateQueryDef (***name*, *sqltext***)**

The **CreateQueryDef** method syntax has these parts.

Part	Description
querydef	An object variable that represents the **QueryDef** object you want to create.
object	An object variable that represents an open **Connection** or **Database** object that will contain the new **QueryDef**.
name	Optional. A **Variant** (**String** subtype) that uniquely names the new **QueryDef**.
sqltext	Optional. A **Variant** (**String** subtype) that is an SQL statement defining the **QueryDef**. If you omit this argument, you can define the **QueryDef** by setting its **SQL** property before or after you append it to a collection.

Remarks

In a Microsoft Jet workspace, if you provide anything other than a zero-length string for the name when you create a **QueryDef**, the resulting **QueryDef** object is automatically appended to the **QueryDefs** collection. In an ODBCDirect workspace, **QueryDef** objects are always temporary.

In an ODBCDirect workspace, the *sqltext* argument can specify an SQL statement or a Microsoft SQL Server stored procedure and its parameters.

If the object specified by *name* is already a member of the **QueryDefs** collection, a run-time error occurs. You can create a temporary **QueryDef** by using a zero-length string for the *name* argument when you execute the **CreateQueryDef** method. You can also accomplish this by setting the **Name** property of a newly created **QueryDef** to a zero-length string (""). Temporary **QueryDef** objects are useful if you want to repeatedly use dynamic SQL statements without having to create any new permanent objects in the **QueryDefs** collection. You can't append a temporary **QueryDef** to any collection because a zero-length string isn't a valid name for a permanent **QueryDef** object. You can always set the **Name** and **SQL** properties of the newly created **QueryDef** object and subsequently append the **QueryDef** to the **QueryDefs** collection.

To run the SQL statement in a **QueryDef** object, use the **Execute** or **OpenRecordset** method.

Using a **QueryDef** object is the preferred way to perform SQL pass-through queries with ODBC databases.

To remove a **QueryDef** object from a **QueryDefs** collection in a Microsoft Jet database, use the **Delete** method on the collection. For an ODBCDirect database, use the **Close** method on the **QueryDef** object.

CreateRelation Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCreateRelationC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCreateRelationX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCreateRelationA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCreateRelationS"}

Creates a new **Relation** object (Microsoft Jet workspaces only).

Syntax

Set *relation* = *database*.**CreateRelation (***name*, *table*, *foreigntable*, *attributes***)**

The **CreateRelation** method syntax uses these parts.

Part	Description
relation	An object variable that represents the **Relation** object you want to create.
database	An object variable that represents the **Database** object for which you want to create the new **Relation** object.
name	Optional. A **Variant** (**String** subtype) that uniquely names the new **Relation** object. See the **Name** property for details on valid **Relation** names.
table	Optional. A **Variant** (**String** subtype) that names the primary table in the relation. If the table doesn't exist before you append the **Relation** object, a run-time error occurs.
foreigntable	Optional. A **Variant** (**String** subtype) that names the foreign table in the relation. If the table doesn't exist before you append the **Relation** object, a run-time error occurs.
attributes	Optional. A constant or combination of constants that contains information about the relationship type. See the **Attributes** property for details.

Remarks

The **Relation** object provides information to the Microsoft Jet database engine about the relationship between fields in two **TableDef** or **QueryDef** objects. You can implement referential integrity by using the **Attributes** property.

If you omit one or more of the optional parts when you use the **CreateRelation** method, you can use an appropriate assignment statement to set or reset the corresponding property before you append the new object to a collection. After you append the object, you can't alter any of its property settings. See the individual property topics for more details.

Before you can use the **Append** method on a **Relation** object, you must append the appropriate **Field** objects to define the primary and foreign key relationship tables.

If *name* refers to an object that is already a member of the collection or if the **Field** object names provided in the subordinate **Fields** collection are invalid, a run-time error occurs when you use the **Append** method.

You can't establish or maintain a relationship between a replicated table and a local table.

To remove a **Relation** object from the **Relations** collection, use the **Delete** method on the collection.

CreateTableDef Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCreateTableDefC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCreateTableDefX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCreateTableDefA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCreateTableDefS"}

Creates a new **TableDef** object (Microsoft Jet workspaces only).

Syntax

Set *tabledef* = *database***.CreateTableDef (***name*, *attributes*, *source*, *connect***)**

The **CreateTableDef** method syntax has these parts.

Part	Description
tabledef	An object variable that represents the **TableDef** object you want to create.
database	An object variable that represents the **Database** object you want to use to create the new **TableDef** object.
name	Optional. A **Variant** (**String** subtype) that uniquely names the new **TableDef** object. See the **Name** property for details on valid **TableDef** names.
attributes	Optional. A constant or combination of constants that indicates one or more characteristics of the new **TableDef** object. See the **Attributes** property for more information.
source	Optional. A **Variant** (**String** subtype) containing the name of a table in an external database that is the original source of the data. The *source* string becomes the **SourceTableName** property setting of the new **TableDef** object.
connect	Optional. A **Variant** (**String** subtype) containing information about the source of an open database, a database used in a pass-through query, or a linked table. See the **Connect** property for more information about valid connection strings.

Remarks

If you omit one or more of the optional parts when you use the **CreateTableDef** method, you can use an appropriate assignment statement to set or reset the corresponding property before you append the new object to a collection. After you append the object, you can alter some but not all of its properties. See the individual property topics for more details.

If *name* refers to an object that is already a member of the collection, or you specify an invalid property in the **TableDef** or **Field** object you're appending, a run-time error occurs when you use the **Append** method. Also, you can't append a **TableDef** object to the **TableDefs** collection until you define at least one **Field** for the **TableDef** object.

To remove a **TableDef** object from the **TableDefs** collection, use the **Delete** method on the collection.

CreateUser Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCreateUserC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCreateUserX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCreateUserA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCreateUserS"}

Creates a new **User** object (Microsoft Jet workspaces only).

Syntax

Set *user* = *object***.CreateUser (***name*, *pid*, *password***)**

The **CreateUser** method syntax has these parts.

Part	Description
user	An object variable that represents the **User** object you want to create.
object	An object variable that represents the **Group** or **Workspace** object for which you want to create the new **User** object.
name	Optional. A **Variant** (**String** subtype) that uniquely names the new **User** object. See the **Name** property for details on valid **User** names.
pid	Optional. A **Variant** (**String** subtype) containing the PID of a user account. The identifier must contain from 4 to 20 alphanumeric characters. See the **PID** property for more information on valid personal identifiers.
password	Optional. A **Variant** (**String** subtype) containing the password for the new **User** object. The password can be up to 14 characters long and can include any characters except the ASCII character 0 (null). See the **Password** property for more information on valid passwords.

Remarks

If you omit one or more of the optional parts when you use the **CreateUser** method, you can use an appropriate assignment statement to set or reset the corresponding property before you append the new object to a collection. After you append the object, you can alter some but not all of its property settings. See the **PID**, **Name**, and **Password** property topics for more details.

If *name* refers to an object that is already a member of the collection, a run-time error occurs when you use the **Append** method.

To remove a **User** object from the **Users** collection, use the **Delete** method on the collection.

CreateWorkspace Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCreateWorkspaceC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCreateWorkspaceX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCreateWorkspaceA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCreateWorkspaceS"}

Creates a new **Workspace** object.

Syntax

Set *workspace* = **CreateWorkspace(***name*, *user*, *password*, *type***)**

The **CreateWorkspace** method syntax has these parts.

Part	Description
workspace	An object variable that represents the **Workspace** object you want to create.
name	A **String** that uniquely names the new **Workspace** object. See the **Name** property for details on valid **Workspace** names.
user	A **String** that identifies the owner of the new **Workspace** object. See the **UserName** property for more information.
password	A **String** containing the password for the new **Workspace** object. The password can be up to 14 characters long and can include any characters except ASCII character 0 (null). See the **Password** property for more information on valid passwords.
type	Optional. A constant that indicates the type of workspace, as described in Settings.

Settings

You can use the following constants for *type*.

Constant	Description
dbUseJet	Creates a Microsoft Jet workspace.
dbUseODBC	Creates an ODBCDirect workspace.

Remarks

Once you use the **CreateWorkspace** method to create a new **Workspace** object, a **Workspace** session is started, and you can refer to the **Workspace** object in your application.

Workspace objects aren't permanent, and you can't save them to disk. Once you create a **Workspace** object, you can't alter any of its property settings, except for the **Name** property, which you can modify before appending the **Workspace** object to the **Workspaces** collection.

You don't have to append the new **Workspace** object to a collection before you can use it. You append a newly created **Workspace** object only if you need to refer to it through the **Workspaces** collection.

The *type* option determines whether the new **Workspace** is a Microsoft Jet or ODBCDirect workspace. If you set *type* to **dbUseODBC** and you haven't already created any Microsoft Jet workspaces, then the Microsoft Jet database engine will not be loaded into memory, and all activity will occur with the ODBC data source subsequently identified in a **Connection** object. If you omit *type*, the **DefaultType** property of **DBEngine** will determine which type of data source the **Workspace** is connected to. You can have both Microsoft Jet and ODBCDirect workspaces open at the same time.

To remove a **Workspace** object from the **Workspaces** collection, close all open databases and connections and then use the **Close** method on the **Workspace** object.

Delete Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthdeleteC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthDeleteX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthDeleteA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthDeleteS"}

- **Recordset** objects — deletes the current record in an updatable **Recordset** object. For ODBCDirect workspaces, the type of driver determines whether **Recordset** objects are updatable and therefore support the **Delete** method.
- Collections — deletes a persistent object from a collection.

Syntax

recordset.**Delete**
collection.**Delete** *objectname*

The **Delete** method syntax has these parts.

Part	Description
recordset	An object variable that represents an updatable **Recordset** object containing the record you want to delete.
collection	An object variable that represents a collection from which you are deleting *objectname*.
objectname	A **String** that is the **Name** property setting of an object in *collection*.

Remarks

You can use the **Delete** method to delete a current record from a **Recordset** or a member from a collection, such as a stored table from a database, a stored field from a table, or a stored index from a table.

Recordsets

A **Recordset** must contain a current record before you use **Delete**; otherwise, a run-time error occurs.

In an updatable **Recordset** object, **Delete** removes the current record and makes it inaccessible. Although you can't edit or use the deleted record, it remains current. Once you move to another record, however, you can't make the deleted record current again. Subsequent references to a deleted record in a **Recordset** are invalid and produce an error.

You can undo a record deletion if you use transactions and the **Rollback** method.

If the base table is the primary table in a cascading delete relationship, deleting the current record may also delete one or more records in a foreign table.

Note To add, edit, or delete a record, there must be a unique index on the record in the underlying data source. If not, a "Permission denied" error will occur on the **AddNew**, **Delete**, or **Edit** method call in a Microsoft Jet workspace, or an "Invalid argument" error will occur on the **Update** method call in an ODBCDirect workspace.

Collections

You can use the **Delete** method to delete a persistent object. However, if the collection is a **Databases**, **Recordsets**, or **Workspaces** collection (each of which is stored only in memory), you can remove an open or active object only by closing that object with the **Close** method.

The deletion of a stored object occurs immediately, but you should use the **Refresh** method on any other collections that may be affected by changes to the database structure.

When you delete a **TableDef** object from the **TableDefs** collection, you delete the table definition and the data in the table.

The following table lists some limitations of the **Delete** method. The object in the first column contains the collection in the second column. The third column indicates if you can delete an object from that collection (for example, you can never delete a **Container** object from the **Containers** collection of a **Database** object).

Object	Collection	Can you use the Delete method?
DBEngine	**Workspaces**	No. Closing the objects deletes them.
DBEngine	**Errors**	No
Workspace	**Connections**	No. Closing the objects deletes them.
Workspace	**Databases**	No. Closing the objects deletes them.
Workspace	**Groups**	Yes
Workspace	**Users**	Yes
Connection	**QueryDefs**	No
Connection	**Recordsets**	No. Closing the objects deletes them.
Database	**Containers**	No
Database	**QueryDefs**	Yes
Database	**Recordsets**	No. Closing the objects deletes them.
Database	**Relations**	Yes
Database	**TableDefs**	Yes
Group	**Users**	Yes
User	**Groups**	Yes
Container	**Documents**	No
QueryDef	**Fields**	No
QueryDef	**Parameters**	No
Recordset	**Fields**	No
Relation	**Fields**	Only when the **Relation** object is a new, unappended object.
TableDef	**Fields**	Only when the **TableDef** object is new and hasn't been appended to the database, or when the **Updatable** property of the **TableDef** is set to **True**.
TableDef	**Indexes**	Only when the **TableDef** object is new and hasn't been appended to the

		database, or when the **Updatable** property of the **TableDef** is set to **True**.
Index	**Fields**	Only when the **Index** object is new and hasn't been appended to the database.
Database, **Field, Index**, **QueryDef**, **TableDef**	**Properties**	Only when the property is user-defined.
DBEngine, **Parameter**, **Recordset**, **Workspace**	**Properties**	No

Edit Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthEditC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthEditX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthEditA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthEditS"}

Copies the current record from an updatable **Recordset** object to the copy buffer for subsequent editing.

Syntax

recordset.**Edit**

The *recordset* placeholder represents an open, updatable **Recordset** object that contains the record you want to edit.

Remarks

Once you use the **Edit** method, changes made to the current record's fields are copied to the copy buffer. After you make the desired changes to the record, use the **Update** method to save your changes.

The current record remains current after you use **Edit**.

Caution If you edit a record and then perform any operation that moves to another record, but without first using **Update**, your changes are lost without warning. In addition, if you close *recordset* or end the procedure which declares the **Recordset** or the parent **Database** or **Connection** object, your edited record is discarded without warning.

Using **Edit** produces an error if:

- There is no current record.
- The **Connection**, **Database**, or **Recordset** object was opened as read-only.
- No fields in the record are updatable.
- The **Database** or **Recordset** was opened for exclusive use by another user (Microsoft Jet workspace).
- Another user has locked the page containing your record (Microsoft Jet workspace).

In a Microsoft Jet workspace, when the **Recordset** object's **LockEdits** property setting is **True** (pessimistically locked) in a multiuser environment, the record remains locked from the time **Edit** is used until the update is complete. If the **LockEdits** property setting is **False** (optimistically locked), the record is locked and compared with the pre-edited record just before it's updated in the database. If the record has changed since you used the **Edit** method, the **Update** operation fails with a run-time error if you use **OpenRecordset** without specifying **dbSeeChanges**. By default, Microsoft Jet-connected ODBC and installable ISAM databases always use optimistic locking.

In an ODBCDirect workspace, once you edit (and use **Update** to update) a record's primary key field, you can no longer edit fields in that record until you close the **Recordset**, and then retrieve the record again in a subsequent query.

Note To add, edit, or delete a record, there must be a unique index on the record in the underlying data source. If not, a "Permission denied" error will occur on the **AddNew**, **Delete**, or **Edit** method call in a Microsoft Jet workspace, or an "Invalid argument" error will occur on the **Update** call in an ODBCDirect workspace.

Execute Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthExecuteC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthExecuteX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthExecuteA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthExecuteS"}

Runs an action query or executes an SQL statement on a specified **Connection** or **Database** object.

Syntax

object.**Execute** *source*, *options*
querydef.**Execute** *options*

The **Execute** method syntax has these parts.

Part	Description
object	A **Connection** or **Database** object variable on which the query will run.
querydef	An object variable that represents the **QueryDef** object whose **SQL** property setting specifies the SQL statement to execute.
source	A **String** that is an SQL statement or the **Name** property value of a **QueryDef** object.
options	Optional. A constant or combination of constants that determines the data integrity characteristics of the query, as specified in Settings.

Settings

You can use the following constants for *options*.

Constant	Description
dbDenyWrite	Denies write permission to other users (Microsoft Jet workspaces only).
dbInconsistent	(Default) Executes inconsistent updates (Microsoft Jet workspaces only).
dbConsistent	Executes consistent updates (Microsoft Jet workspaces only).
dbSQLPassThrough	Executes an SQL pass-through query. Setting this option passes the SQL statement to an ODBC database for processing (Microsoft Jet workspaces only).
dbFailOnError	Rolls back updates if an error occurs (Microsoft Jet workspaces only).
dbSeeChanges	Generates a run-time error if another user is changing data you are editing (Microsoft Jet workspaces only).
dbRunAsync	Executes the query asynchronously (ODBCDirect **Connection** and **QueryDef** objects only).
dbExecDirect	Executes the statement without first calling **SQLPrepare** ODBC API function (ODBCDirect **Connection** and **QueryDef** objects only).

Note The constants **dbConsistent** and **dbInconsistent** are mutually exclusive. You can use one or the other, but not both in a given instance of **OpenRecordset**. Using both **dbConsistent** and

dbInconsistent causes an error.

Remarks

The **Execute** method is valid only for action queries. If you use **Execute** with another type of query, an error occurs. Because an action query doesn't return any records, **Execute** doesn't return a **Recordset**. (Executing an SQL pass-through query in an ODBCDirect workspace will not return an error if a **Recordset** isn't returned.)

Use the **RecordsAffected** property of the **Connection**, **Database**, or **QueryDef** object to determine the number of records affected by the most recent **Execute** method. For example, **RecordsAffected** contains the number of records deleted, updated, or inserted when executing an action query. When you use the **Execute** method to run a query, the **RecordsAffected** property of the **QueryDef** object is set to the number of records affected.

In a Microsoft Jet workspace, if you provide a syntactically correct SQL statement and have the appropriate permissions, the **Execute** method won't fail — even if not a single row can be modified or deleted. Therefore, always use the **dbFailOnError** option when using the **Execute** method to run an update or delete query. This option generates a run-time error and rolls back all successful changes if any of the records affected are locked and can't be updated or deleted.

For best performance in a Microsoft Jet workspace, especially in a multiuser environment, nest the **Execute** method inside a transaction. Use the **BeginTrans** method on the current **Workspace** object, then use the **Execute** method, and complete the transaction by using the **CommitTrans** method on the **Workspace**. This saves changes on disk and frees any locks placed while the query is running.

In an ODBCDirect workspace, if you include the optional **dbRunAsync** constant, the query runs asynchronously. To determine whether an asynchronous query is still executing, check the value of the **StillExecuting** property on the object from which the **Execute** method was called. To terminate execution of an asynchronous **Execute** method call, use the **Cancel** method.

FillCache Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthFillCacheC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthFillCacheX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthFillCacheA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthFillCacheS"}

Fills all or a part of a local cache for a **Recordset** object that contains data from a Microsoft Jet-connected ODBC data source (Microsoft Jet-connected ODBC databases only).

Syntax

recordset.**FillCache** *rows*, *startbookmark*

The **FillCache** method syntax has these parts.

Part	Description
recordset	An object variable that represents a **Recordset** object created from an ODBC data source, such as a **TableDef** representing a linked table or a **QueryDef** object derived from such a **TableDef**.
rows	Optional. A **Variant** (**Integer** subtype) that specifies the number of rows to store in the cache. If you omit this argument, the value is determined by the **CacheSize** property setting.
startbookmark	Optional. A **Variant** (**String** subtype) that specifies a bookmark. The cache is filled starting from the record indicated by this bookmark. If you omit this argument, the cache is filled starting from the record indicated by the **CacheStart** property.

Remarks

Caching improves the performance of an application that retrieves data from a remote server. A cache is space in local memory that holds the data most recently retrieved from the server; this assumes that the data will probably be requested again while the application is running. When a user requests data, the Microsoft Jet database engine checks the cache for the data first rather than retrieving it from the server, which takes more time. The cache doesn't save data that doesn't come from an ODBC data source.

Rather than waiting for the cache to be filled with records as they are retrieved, you can use the **FillCache** method to explicitly fill the cache at any time. This is a faster way to fill the cache because **FillCache** retrieves several records at once instead of one at a time. For example, while you view each screenful of records, your application uses **FillCache** to retrieve the next screenful of records for viewing.

Any Microsoft Jet-connected ODBC data source that you access with **Recordset** objects can have a local cache. To create the cache, open a **Recordset** object from the remote data source, and then set the **CacheSize** and **CacheStart** properties of the **Recordset**.

If *rows* and *startbookmark* create a range of records that is partially or entirely outside the range of records specified by the **CacheSize** and **CacheStart** properties, the portion of the *recordset* outside this range is ignored and will not be loaded into the cache.

If **FillCache** requests more records than the number remaining in the remote data source, Microsoft Jet retrieves only the remaining records, and no error occurs.

Notes

- Records retrieved from the cache don't reflect concurrent changes that other users made to the source data.

- **FillCache** only retrieves records not already cached. To force an update of all the cached data, set the **CacheSize** property of the **Recordset** to 0, reset it to the size of the cache you originally requested, and then use **FillCache**.

FindFirst, FindLast, FindNext, FindPrevious Methods

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthFindFirstC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthFindFirstX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthFindFirstA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthFindFirstS"}

Locates the first, last, next, or previous record in a dynaset- or snapshot-type **Recordset** object that satisfies the specified criteria and makes that record the current record (Microsoft Jet workspaces only).

Syntax

recordset.{**FindFirst** | **FindLast** | **FindNext** | **FindPrevious**} *criteria*

The **Find** methods have these parts.

Part	Description
recordset	An object variable that represents an existing dynaset- or snapshot-type **Recordset** object.
criteria	A **String** used to locate the record. It is like the WHERE clause in an SQL statement, but without the word WHERE.

Remarks

If you want to include all the records in your search — not just those that meet a specific condition — use the Move methods to move from record to record. To locate a record in a table-type **Recordset**, use the **Seek** method.

If a record matching the criteria isn't located, the current record pointer is unknown, and the **NoMatch** property is set to **True**. If *recordset* contains more than one record that satisfies the criteria, **FindFirst** locates the first occurrence, **FindNext** locates the next occurrence, and so on.

Each of the **Find** methods begins its search from the location and in the direction specified in the following table.

Find method	Begins searching at	Search direction
FindFirst	Beginning of recordset	End of recordset
FindLast	End of recordset	Beginning of recordset
FindNext	Current record	End of recordset
FindPrevious	Current record	Beginning of recordset

When you use the **FindLast** method, the Microsoft Jet database engine fully populates your **Recordset** before beginning the search, if this hasn't already happened.

Using one of the **Find** methods isn't the same as using a **Move** method, however, which simply makes the first, last, next, or previous record current without specifying a condition. You can follow a Find operation with a Move operation.

Always check the value of the **NoMatch** property to determine whether the Find operation has succeeded. If the search succeeds, **NoMatch** is **False**. If it fails, **NoMatch** is **True** and the current record isn't defined. In this case, you must position the current record pointer back to a valid record.

Using the **Find** methods with Microsoft Jet-connected ODBC-accessed recordsets can be inefficient. You may find that rephrasing your *criteria* to locate a specific record is faster, especially when working with large recordsets.

In an ODBCDirect workspace, the **Find** and **Seek** methods are not available on any type of **Recordset** object, because executing a **Find** or **Seek** through an ODBC connection is not very efficient over the network. Instead, you should design the query (that is, using the *source* argument to

the **OpenRecordset** method) with an appropriate WHERE clause that restricts the returned records to only those that meet the criteria you would otherwise use in a **Find** or **Seek** method.

When working with Microsoft Jet-connected ODBC databases and large dynaset-type **Recordset** objects, you might discover that using the **Find** methods or using the **Sort** or **Filter** property is slow. To improve performance, use SQL queries with customized ORDER BY or WHERE clauses, parameter queries, or **QueryDef** objects that retrieve specific indexed records.

You should use the U.S. date format (month-day-year) when you search for fields containing dates, even if you're not using the U.S. version of the Microsoft Jet database engine; otherwise, the data may not be found. Use the Visual Basic **Format** function to convert the date. For example:

```
rstEmployees.FindFirst "HireDate > #" _
   & Format(mydate, 'm-d-yy' ) & "#"
```

If *criteria* is composed of a string concatenated with a non-integer value, and the system parameters specify a non-U.S. decimal character such as a comma (for example, `strSQL = "PRICE > " & lngPrice`, and `lngPrice = 125,50`), an error occurs when you try to call the method. This is because during concatenation, the number will be converted to a string using your system's default decimal character, and Microsoft Jet SQL only accepts U.S. decimal characters.

Notes

- For best performance, the *criteria* should be in either the form "*field* = *value*" where *field* is an indexed field in the underlying base table, or "*field* LIKE *prefix*" where *field* is an indexed field in the underlying base table and *prefix* is a prefix search string (for example, `"ART*"`).
- In general, for equivalent types of searches, the **Seek** method provides better performance than the **Find** methods. This assumes that table-type **Recordset** objects alone can satisfy your needs.

GetChunk Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthGetChunkC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthGetChunkX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthGetChunkA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthGetChunkS"}

Returns all or a portion of the contents of a Memo or Long Binary **Field** object in the **Fields** collection of a **Recordset** object.

Syntax

Set *variable* = *recordset* **!** *field***.GetChunk (***offset*, *numbytes***)**

The **GetChunk** method syntax has these parts.

Part	Description
variable	A **Variant** (**String** subtype) that receives the data from the **Field** object named by *field.*
recordset	An object variable that represents the **Recordset** object containing the **Fields** collection.
field	An object variable that represents a **Field** object whose **Type** property is set to **dbMemo** (Memo) or **dbLongBinary** (Long Binary).
offset	A **Long** value equal to the number of bytes to skip before copying begins.
numbytes	A **Long** value equal to the number of bytes you want to return.

Remarks

The bytes returned by **GetChunk** are assigned to *variable*. Use **GetChunk** to return a portion of the total data value at a time. You can use the **AppendChunk** method to reassemble the pieces.

If *offset* is 0, **GetChunk** begins copying from the first byte of the field.

If *numbytes* is greater than the number of bytes in the field, **GetChunk** returns the actual number of remaining bytes in the field.

Caution Use a Memo field for text, and put binary data only in Long Binary fields. Doing otherwise will cause undesirable results.

GetRows Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthGetRowsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthGetRowsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthGetRowsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthGetRowsS"}

Retrieves multiple rows from a **Recordset** object.

Syntax

Set *varArray* = *recordset*.**GetRows (***numrows***)**

The **GetRows** method syntax has the following parts.

Part	Description
varArray	A **Variant** that stores the returned data.
recordset	An object variable that represents a **Recordset** object.
numrows	A **Variant** that is equal to the number of rows to retrieve.

Remarks

Use the **GetRows** method to copy records from a **Recordset**. **GetRows** returns a two-dimensional array. The first subscript identifies the field and the second identifies the row number. For example, `intField` represents the field, and `intRecord` identifies the row number:

```
avarRecords(intField, intRecord)
```

To get the first field value in the second row returned, use code like the following:

```
field1 = avarRecords(0,1)
```

To get the second field value in the first row, use code like the following:

```
field2 = avarRecords(1,0)
```

The `avarRecords` variable automatically becomes a two-dimensional array when **GetRows** returns data.

If you request more rows than are available, then **GetRows** returns only the number of available rows. You can use the Visual Basic for Applications **UBound** function to determine how many rows **GetRows** actually retrieved, because the array is sized to fit the number of returned rows. For example, if you returned the results into a **Variant** called `varA`, you could use the following code to determine how many rows were actually returned:

```
numReturned = UBound(varA,2) + 1
```

You need to use `"+ 1"` because the first row returned is in the 0 element of the array. The number of rows that you can retrieve is constrained by the amount of available memory. You shouldn't use **GetRows** to retrieve an entire table into an array if it is large.

Because **GetRows** returns all fields of the **Recordset** into the array, including Memo and Long Binary fields, you might want to use a query that restricts the fields returned.

After you call **GetRows**, the current record is positioned at the next unread row. That is, **GetRows** has the same effect on the current record as **Move** *numrows*.

If you are trying to retrieve all the rows by using multiple **GetRows** calls, use the **EOF** property to be sure that you're at the end of the **Recordset**. **GetRows** returns less than the number requested if it's at the end of the **Recordset**, or if it can't retrieve a row in the range requested. For example, if you're trying to retrieve 10 records, but you can't retrieve the fifth record, **GetRows** returns four records and makes the fifth record the current record. This will not generate a run-time error. This might occur if another user deletes a record in a dynaset-type **Recordset**. See the example for a demonstration of

how to handle this.

Idle Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthIdleC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthIdleX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthIdleA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthIdleS"}

Suspends data processing, enabling the Microsoft Jet database engine to complete any pending tasks, such as memory optimization or page timeouts (Microsoft Jet workspaces only).

Syntax

DBEngine.Idle [**dbRefreshCache**]

Remarks

The **Idle** method allows the Microsoft Jet database engine to perform background tasks that may not be up-to-date because of intense data processing. This is often true in multiuser, multitasking environments that don't have enough background processing time to keep all records in a **Recordset** current.

Usually, read locks are removed and data in local dynaset-type **Recordset** objects are updated only when no other actions (including mouse movements) occur. If you periodically use the **Idle** method, Microsoft Jet can catch up on background processing tasks by releasing unneeded read locks.

Specifying the optional **dbRefreshCache** argument forces any pending writes to .mdb files, and refreshes memory with the most current data from the .mdb file.

You don't need to use this method in single-user environments unless multiple instances of an application are running. The **Idle** method may increase performance in a multiuser environment because it forces the database engine to write data to disk, releasing locks on memory.

Note You can also release read locks by making operations part of a transaction.

MakeReplica Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthMakeReplicaC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthMakeReplicaX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthMakeReplicaA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthMakeReplicaS"}

Makes a new replica from another database replica (Microsoft Jet workspaces only).

Syntax

database.**MakeReplica** *replica*, *description*, *options*

The **MakeReplica** method syntax has the following parts.

Part	Description
database	An object variable that represents an existing **Database** that is a replica.
replica	A **String** that is the path and file name of the new replica. If *replica* is an existing file name, then an error occurs.
description	A **String** that describes the replica that you are creating.
options	Optional. A constant or combination of constants that specifies characteristics of the replica you are creating, as specified in Settings.

Settings

You can use one or more of the following constants in the *options* argument.

Constant	Description
dbRepMakePartial	Creates a partial replica.
dbRepMakeReadOnly	Prevents users from modifying the replicable objects of the new replica; however, when you synchronize the new replica with another member of the replica set, design and data changes will be propagated to the new replica.

Remarks

A newly created partial replica will have all **ReplicaFilter** properties set to **False**, meaning that no data will be in the tables.

Move Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthMoveC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthMoveX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthMoveA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthMoveS"}

Moves the position of the current record in a **Recordset** object.

Syntax

recordset.**Move** *rows*, *start*

The **Move** method syntax has these parts.

Part	Description
recordset	An object variable that represents the **Recordset** object whose current record position is being moved.
rows	A signed **Long** value specifying the number of rows the position will move. If *rows* is greater than 0, the position is moved forward (toward the end of the file). If *rows* is less than 0, the position is moved backward (toward the beginning of the file).
startbookmark	Optional. A **Variant** (**String** subtype) value identifying a bookmark. If you specify *startbookmark,* the move begins relative to this bookmark. Otherwise, **Move** begins from the current record.

Remarks

If you use **Move** to position the current record pointer before the first record, the current record pointer moves to the beginning of the file. If the **Recordset** contains no records and its **BOF** property is **True**, using this method to move backward causes an error.

If you use **Move** to position the current record pointer after the last record, the current record pointer position moves to the end of the file. If the **Recordset** contains no records and its **EOF** property is **True**, then using this method to move forward causes an error.

If either the **BOF** or **EOF** property is **True** and you attempt to use the **Move** method without a valid bookmark, a run-time error occurs.

Notes

- When you use **Move** on a forward-only-type **Recordset** object, the *rows* argument must be a positive integer and bookmarks aren't allowed. This means you can only move forward.
- To make the first, last, next, or previous record in a **Recordset** the current record, use either the **MoveFirst, MoveLast, MoveNext, or MovePrevious** method.
- Using **Move** with *rows* equal to 0 is an easy way to retrieve the underlying data for the current record. This is useful if you want to make sure that the current record has the most recent data from the base tables. It will also cancel any pending **Edit** or **AddNew** calls.

MoveFirst, MoveLast, MoveNext, MovePrevious Methods

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthMoveFirstC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthMoveFirstX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthMoveFirstA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthMoveFirstS"}

Move to the first, last, next, or previous record in a specified **Recordset** object and make that record the current record.

Syntax

recordset.{**MoveFirst** | **MoveLast** [**dbRunAsync**] | **MoveNext** | **MovePrevious**}

The *recordset* placeholder is an object variable that represents an open **Recordset** object.

Remarks

Use the **Move** methods to move from record to record without applying a condition.

Caution If you edit the current record, be sure you use the **Update** method to save the changes before you move to another record. If you move to another record without updating, your changes are lost without warning.

When you open a **Recordset**, the first record is current and the **BOF** property is **False**. If the **Recordset** contains no records, the **BOF** property is **True**, and there is no current record.

If the first or last record is already current when you use **MoveFirst** or **MoveLast**, the current record doesn't change.

If you use **MovePrevious** when the first record is current, the **BOF** property is **True**, and there is no current record. If you use **MovePrevious** again, an error occurs, and **BOF** remains **True**.

If you use **MoveNext** when the last record is current, the **EOF** property is **True**, and there is no current record. If you use **MoveNext** again, an error occurs, and **EOF** remains **True**.

If *recordset* refers to a table-type **Recordset** (Microsoft Jet workspaces only), movement follows the current index. You can set the current index by using the **Index** property. If you don't set the current index, the order of returned records is undefined.

Important You can use the **MoveLast** method to fully populate a dynaset- or snapshot-type **Recordset** to provide the current number of records in the **Recordset**. However, if you use **MoveLast** in this way, you can slow down your application's performance. You should only use **MoveLast** to get a record count if it is absolutely necessary to obtain an accurate record count on a newly opened **Recordset**. If you use the **dbRunAsync** constant with **MoveLast**, the method call is asynchronous. You can use the **StillExecuting** property to determine when the **Recordset** is fully populated, and you can use the **Cancel** method to terminate execution of the asynchronous **MoveLast** method call.

You can't use the **MoveFirst**, **MoveLast**, and **MovePrevious** methods on a forward-only–type **Recordset** object.

To move the position of the current record in a **Recordset** object a specific number of records forward or backward, use the **Move** method.

NewPassword Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthNewPasswordC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthNewPasswordX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthNewPasswordA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthNewPasswordS"}

Changes the password of an existing user account or Microsoft Jet database (Microsoft Jet workspaces only).

Syntax

object.**NewPassword** *oldpassword*, *newpassword*

The **NewPassword** method syntax has these parts.

Part	Description
object	An object variable that represents the **User** object or a Microsoft Jet 3.*x* **Database** object whose **Password** property you want to change.
oldpassword	A **String** that is the current setting of the **Password** property of the **User** or Jet 3.*x* **Database** object.
newpassword	A **String** that is the new setting of the **Password** property of the **User** or Jet 3.*x* **Database** object.

Remarks

The *oldpassword* and *newpassword* strings can be up to 14 characters long and can include any characters except the ASCII character 0 (null). To clear the password, use a zero-length string ("") for *newpassword*.

Passwords are case-sensitive.

If *object* refers to a **User** object that is not yet appended to a **Users** collection, an error occurs. To set a new password, you must either log on as the user whose account you're changing, or you must be a member of the Admins group. The **Password** property of a **User** object is write-only — users can't read the current value.

If *object* refers to a Microsoft Jet version 3.0 or later **Database** object, this method offers some security by means of password protection. When you create or open a Microsoft Jet 3.*x* .mdb file, part of the **Connect** connection string can describe the password.

If a database has no password, Microsoft Jet will automatically create one by passing a zero-length string ("") for the old password.

Caution If you lose your password, you can never open the database again.

OpenDatabase Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthOpenDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthOpenDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthOpenDatabaseA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthOpenDatabaseS"}

Opens a specified database in a **Workspace** object and returns a reference to the **Database** object that represents it.

Syntax

Set *database* = *workspace*.**OpenDatabase (***dbname*, *options*, *read-only*, *connect***)**

The **OpenDatabase** method syntax has these parts.

Part	Description
database	An object variable that represents the **Database** object that you want to open.
workspace	Optional. An object variable that represents the existing **Workspace** object that will contain the database. If you don't include a value for *workspace*, **OpenDatabase** uses the default workspace.
dbname	A **String** that is the name of an existing Microsoft Jet database file, or the data source name (DSN) of an ODBC data source. See the **Name** property for more information about setting this value.
options	Optional. A **Variant** that sets various options for the database, as specified in Settings.
read-only	Optional. A **Variant** (**Boolean** subtype) value that is **True** if you want to open the database with read-only access, or **False** (default) if you want to open the database with read/write access.
connect	Optional. A **Variant** (**String** subtype) that specifies various connection information, including passwords.

Settings

For Microsoft Jet workspaces, you can use the following values for the *options* argument.

Setting	Description
True	Opens the database in exclusive mode.
False	(Default) Opens the database in shared mode.

For ODBCDirect workspaces, the *options* argument determines if and when to prompt the user to establish the connection. You can use one of the following constants.

Constant	Description
dbDriverNoPrompt	The ODBC Driver Manager uses the connection string provided in *dbname* and *connect*. If you don't provide sufficient information, a run-time error occurs.
dbDriverPrompt	The ODBC Driver Manager displays the **ODBC Data Sources** dialog box, which displays any relevant information supplied in *dbname* or *connect*. The connection string is made up of the

	DSN that the user selects via the dialog boxes, or, if the user doesn't specify a DSN, the default DSN is used.
dbDriverComplete	(Default) If the *connect* and *dbname* arguments include all the necessary information to complete a connection, the ODBC Driver Manager uses the string in *connect*. Otherwise it behaves as it does when you specify **dbDriverPrompt**.
dbDriverCompleteRequired	This option behaves like **dbDriverComplete** except the ODBC driver disables the prompts for any information not required to complete the connection.

Remarks

When you open a database, it is automatically added to the **Databases** collection. Further, in an ODBCDirect workspace, the **Connection** object corresponding to the new **Database** object is also created and appended to the **Connections** collection of the same **Workspace** object.

Some considerations apply when you use *dbname*:

- If it refers to a database that is already open for exclusive access by another user, an error occurs.
- If it doesn't refer to an existing database or valid ODBC data source name, an error occurs.
- If it's a zero-length string ("") and *connect* is `"ODBC;"`, a dialog box listing all registered ODBC data source names is displayed so the user can select a database.
- If you're opening a database through an ODBCDirect workspace and you provide the DSN in *connect*, you can set *dbname* to a string of your choice that you can use to reference this database in subsequent code.

The *connect* argument is expressed in two parts: the database type, followed by a semicolon (;) and the optional arguments. You must first provide the database type, such as `"ODBC;"` or `"FoxPro 2.5;"`. The optional arguments follow in no particular order, separated by semicolons. One of the parameters may be the password (if one is assigned). For example:

```
"FoxPro 2.5; pwd=mypassword"
```

Using the **NewPassword** method on a **Database** object other than an ODBCDirect database changes the password parameter that appears in the `";pwd=..."` part of this argument. You must supply the *options* and *read-only* arguments to supply a source string. See the **Connect** property for syntax.

To close a database, and thus remove the **Database** object from the **Databases** collection, use the **Close** method on the object.

Note When you access a Microsoft Jet-connected ODBC data source, you can improve your application's performance by opening a **Database** object connected to the ODBC data source, rather than by linking individual **TableDef** objects to specific tables in the ODBC data source.

OpenRecordset Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthOpenRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthOpenRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthOpenRecordsetA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthOpenRecordsetS"}

Creates a new **Recordset** object and appends it to the **Recordsets** collection.

Syntax

For **Connection** and **Database** objects:

Set *recordset* = *object***.OpenRecordset (***source***,** *type***,** *options***,** *lockedits***)**

For **QueryDef**, **Recordset**, and **TableDef** objects:

Set *recordset* = *object***.OpenRecordset (***type***,** *options***,** *lockedits***)**

The **OpenRecordset** method syntax has these parts.

Part	Description
recordset	An object variable that represents the **Recordset** object you want to open.
object	An object variable that represents an existing object from which you want to create the new **Recordset**.
source	A **String** specifying the source of the records for the new **Recordset**. The source can be a table name, a query name, or an SQL statement that returns records. For table-type **Recordset** objects in Microsoft Jet databases, the source can only be a table name.
type	Optional. A constant that indicates the type of **Recordset** to open, as specified in Settings.
options	Optional. A combination of constants that specify characteristics of the new **Recordset**, as listed in Settings.
lockedits	Optional. A constant that determines the locking for the **Recordset**, as specified in Settings.

Settings

You can use one of the following constants for the *type* argument.

Constant	Description
dbOpenTable	Opens a table-type **Recordset** object (Microsoft Jet workspaces only).
dbOpenDynamic	Opens a dynamic-type **Recordset** object, which is similar to an ODBC dynamic cursor. (ODBCDirect workspaces only)
dbOpenDynaset	Opens a dynaset-type **Recordset** object, which is similar to an ODBC keyset cursor.
dbOpenSnapshot	Opens a snapshot-type **Recordset** object, which is similar to an ODBC static cursor.
dbOpenForwardOnly	Opens a forward-only-type **Recordset** object.

Note If you open a **Recordset** in a Microsoft Jet workspace and you don't specify a type,

OpenRecordset creates a table-type **Recordset**, if possible. If you specify a linked table or query, **OpenRecordset** creates a dynaset-type **Recordset**. In an ODBCDirect workspace, the default setting is **dbOpenForwardOnly**.

You can use a combination of the following constants for the *options* argument.

Constant	Description
dbAppendOnly	Allows users to append new records to the **Recordset**, but prevents them from editing or deleting existing records (Microsoft Jet dynaset-type **Recordset** only).
dbSQLPassThrough	Passes an SQL statement to a Microsoft Jet-connected ODBC data source for processing (Microsoft Jet snapshot-type **Recordset** only).
dbSeeChanges	Generates a run-time error if one user is changing data that another user is editing (Microsoft Jet dynaset-type **Recordset** only). This is useful in applications where multiple users have simultaneous read/write access to the same data.
dbDenyWrite	Prevents other users from modifying or adding records (Microsoft Jet **Recordset** objects only).
dbDenyRead	Prevents other users from reading data in a table (Microsoft Jet table-type **Recordset** only).
dbForwardOnly	Creates a forward-only **Recordset** (Microsoft Jet snapshot-type **Recordset** only). It is provided only for backward compatibility, and you should use the **dbOpenForwardOnly** constant in the *type* argument instead of using this option.
dbReadOnly	Prevents users from making changes to the **Recordset** (Microsoft Jet only). The **dbReadOnly** constant in the *lockedits* argument replaces this option, which is provided only for backward compatibility.
dbRunAsync	Runs an asynchronous query (ODBCDirect workspaces only).
dbExecDirect	Runs a query by skipping **SQLPrepare** and directly calling **SQLExecDirect** (ODBCDirect workspaces only). Use this option only when you're not opening a **Recordset** based on a parameter query. For more information, see the "Microsoft ODBC 3.0 Programmer's Reference."
dbInconsistent	Allows inconsistent updates (Microsoft Jet dynaset-type and snapshot-type **Recordset** objects only).
dbConsistent	Allows only consistent updates (Microsoft Jet dynaset-type and snapshot-type **Recordset** objects only).

Note The constants **dbConsistent** and **dbInconsistent** are mutually exclusive, and using both causes an error. Supplying a *lockedits* argument when *options* uses the **dbReadOnly** constant also causes an error.

You can use the following constants for the *lockedits* argument.

Constant	Description
dbReadOnly	Prevents users from making changes to the **Recordset** (default for ODBCDirect workspaces). You can use **dbReadOnly** in either the *options* argument or the *lockedits* argument, but not both. If you use it for both arguments, a run-time error occurs.
dbPessimistic	Uses pessimistic locking to determine how changes are made to the **Recordset** in a multiuser environment. The page containing the record you're editing is locked as soon as you use the **Edit** method (default for Microsoft Jet workspaces).
dbOptimistic	Uses optimistic locking to determine how changes are made to the **Recordset** in a multiuser environment. The page containing the record is not locked until the **Update** method is executed.
dbOptimisticValue	Uses optimistic concurrency based on row values (ODBCDirect workspaces only).
dbOptimisticBatch	Enables batch optimistic updating (ODBCDirect workspaces only).

Remarks

In a Microsoft Jet workspace, if *object* refers to a **QueryDef** object, or a dynaset- or snapshot-type **Recordset**, or if *source* refers to an SQL statement or a **TableDef** that represents a linked table, you can't use **dbOpenTable** for the *type* argument; if you do, a run-time error occurs. If you want to use an SQL pass-through query on a linked table in a Microsoft Jet-connected ODBC data source, you must first set the **Connect** property of the linked table's database to a valid ODBC connection string. If you only need to make a single pass through a **Recordset** opened from a Microsoft Jet-connected ODBC data source, you can improve performance by using **dbOpenForwardOnly** for the *type* argument.

If *object* refers to a dynaset- or snapshot-type **Recordset**, the new **Recordset** is of the same type *object*. If *object* refers to a table-type **Recordset** object, the type of the new object is a dynaset-type **Recordset**. You can't open new **Recordset** objects from forward-only–type or ODBCDirect **Recordset** objects.

In an ODBCDirect workspace, you can open a **Recordset** containing more than one select query in the *source* argument, such as

```
"SELECT LastName, FirstName FROM Authors
WHERE LastName = 'Smith';
SELECT Title, ISBN FROM Titles
WHERE ISBN Like '1-55615-*'"
```

The returned **Recordset** will open with the results of the first query. To obtain the result sets of records from subsequent queries, use the **NextRecordset** method.

Note You can send DAO queries to a variety of different database servers with ODBCDirect, and different servers will recognize slightly different dialects of SQL. Therefore, context-sensitive Help is no longer provided for Microsoft Jet SQL, although online Help for Microsoft Jet SQL is still included through the Help menu. Be sure to check the appropriate reference documentation for the SQL dialect of your database server when using either ODBCDirect connections or pass-through queries in Microsoft Jet-connected client/server applications.

Use the **dbSeeChanges** constant in a Microsoft Jet workspace if you want to trap changes while two or more users are editing or deleting the same record. For example, if two users start editing the same record, the first user to execute the **Update** method succeeds. When the second user invokes the **Update** method, a run-time error occurs. Similarly, if the second user tries to use the **Delete** method to delete the record, and the first user has already changed it, a run-time error occurs.

Typically, if the user gets this error while updating a record, your code should refresh the contents of the fields and retrieve the newly modified values. If the error occurs while deleting a record, your code could display the new record data to the user and a message indicating that the data has recently changed. At this point, your code can request a confirmation that the user still wants to delete the record.

You should also use the **dbSeeChanges** constant if you open a **Recordset** in a Microsoft Jet-connected ODBC workspace against a Microsoft SQL Server 6.0 (or later) table that has an IDENTITY column, otherwise an error may result.

In an ODBCDirect workspace, you can execute asynchronous queries by setting the **dbRunAsync** constant in the *options* argument. This allows your application to continue processing other statements while the query runs in the background. But, you cannot access the **Recordset** data until the query has completed. To determine whether the query has finished executing, check the **StillExecuting** property of the new **Recordset**. If the query takes longer to complete than you anticipated, you can terminate execution of the query with the **Cancel** method.

Opening more than one **Recordset** on an ODBC data source may fail because the connection is busy with a prior **OpenRecordset** call. One way around this is to use a server-side cursor and ODBCDirect, if the server supports this. Another solution is to fully populate the **Recordset** by using the **MoveLast** method as soon as the **Recordset** is opened.

If you open a **Connection** object with **DefaultCursorDriver** set to **dbUseClientBatchCursor**, you can open a **Recordset** to cache changes to the data (known as batch updating) in an ODBCDirect workspace. Include **dbOptimisticBatch** in the *lockedits* argument to enable update caching. See the **Update** method topic for details about how to write changes to disk immediately, or to cache changes and write them to disk as a batch.

Closing a **Recordset** with the **Close** method automatically deletes it from the **Recordsets** collection.

Note If *source* refers to an SQL statement composed of a string concatenated with a non-integer value, and the system parameters specify a non-U.S. decimal character such as a comma (for example, `strSQL = "PRICE > " & lngPrice`, and `lngPrice = 125,50`), an error occurs when you try to open the **Recordset**. This is because during concatenation, the number will be converted to a string using your system's default decimal character, and SQL only accepts U.S. decimal characters.

Refresh Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthRefreshC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthRefreshX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthRefreshA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthRefreshS"}

Updates the objects in a collection to reflect the current database's schema.

Syntax

collection.**Refresh**

The *collection* placeholder is an object variable that represents a persistent collection.

Remarks

You can't use the **Refresh** method with collections that aren't persistent, such as **Connections**, **Databases**, **Recordsets**, **Workspaces**, or the **QueryDefs** collection of a **Connection** object.

To determine the position that the Microsoft Jet database engine uses for **Field** objects in the **Fields** collection of a **QueryDef**, **Recordset**, or **TableDef** object, use the **OrdinalPosition** property of each **Field** object. Changing the **OrdinalPosition** property of a **Field** object may not change the order of the **Field** objects in the collection until you use the **Refresh** method.

Use the **Refresh** method in multiuser environments in which other users may change the database. You may also need to use it on any collections that are indirectly affected by changes to the database. For example, if you change a **Users** collection, you may need to refresh a **Groups** collection before using the **Groups** collection.

A collection is filled with objects the first time it's referred to and won't automatically reflect subsequent changes other users make. If it's likely that another user has changed a collection, use the **Refresh** method on the collection immediately before carrying out any task in your application that assumes the presence or absence of a particular object in the collection. This will ensure that the collection is as up-to-date as possible. On the other hand, using **Refresh** can unnecessarily slow performance.

RefreshLink Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthRefreshLinkC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthRefreshLinkX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthRefreshLinkA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthRefreshLinkS"}

Updates the connection information for a linked table (Microsoft Jet workspaces only).

Syntax

tabledef.**RefreshLink**

The *tabledef* placeholder specifies the **TableDef** object representing the linked table whose connection information you want to update.

Remarks

To change the connection information for a linked table, reset the **Connect** property of the corresponding **TableDef** object and then use the **RefreshLink** method to update the information. Using **RefreshLink** method doesn't change the linked table's properties and **Relation** objects.

For this connection information to exist in all collections associated with the **TableDef** object that represents the linked table, you must use the **Refresh** method on each collection.

RegisterDatabase Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthRegisterDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthRegisterDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthRegisterDatabaseA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthRegisterDatabaseS"}

Enters connection information for an ODBC data source in the Windows Registry. The ODBC driver needs connection information when the ODBC data source is opened during a session.

Syntax

DBEngine.RegisterDatabase *dbname*, *driver*, *silent*, *attributes*

The **RegisterDatabase** method syntax has these parts.

Part	Description
dbname	A **String** that is the name used in the **OpenDatabase** method. It refers to a block of descriptive information about the data source. For example, if the data source is an ODBC remote database, it could be the name of the server.
driver	A **String** that is the name of the ODBC driver. This isn't the name of the ODBC driver DLL file. For example, SQL Server is a driver name, but SQLSRVR.dll is the name of a DLL file. You must have ODBC and the appropriate driver already installed.
silent	A **Boolean** that is **True** if you don't want to display the ODBC driver dialog boxes that prompt for driver-specific information; or **False** if you want to display the ODBC driver dialog boxes. If *silent* is **True**, *attributes* must contain all the necessary driver-specific information or the dialog boxes are displayed anyway.
attributes	A **String** that is a list of keywords to be added to the Windows Registry. The keywords are in a carriage-return–delimited string.

Remarks

If the database is already registered (connection information is already entered) in the Windows Registry when you use the **RegisterDatabase** method, the connection information is updated.

If the **RegisterDatabase** method fails for any reason, no changes are made to the Windows Registry, and an error occurs.

For more information about ODBC drivers such as SQL Server, see the Help file provided with the driver.

You should use the **ODBC Data Sources** dialog box in the Control Panel to add new data sources, or to make changes to existing entries. However, if you use the **RegisterDatabase** method, you should set the *silent* option to **True**.

RepairDatabase Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthRepairDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthRepairDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthRepairDatabaseA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthRepairDatabaseS"}

Attempts to repair a corrupted Microsoft Jet database (Microsoft Jet databases only).

Syntax

DBEngine.RepairDatabase *dbname*

The *dbname* argument is a **String** that is the path and file name for an existing Microsoft Jet database file. If you omit the path, only the current directory is searched. If your system supports the uniform naming convention (UNC), you can also specify a network path, such as `"\\server1\share1\dir1\db1.mdb"`.

Remarks

You must close the database specified by *dbname* before you repair it. In a multiuser environment, other users can't have *dbname* open while you're repairing it. If *dbname* isn't closed or isn't available for exclusive use, an error occurs.

This method attempts to repair a database that was marked as possibly corrupt by an incomplete write operation. This can occur if an application using the Microsoft Jet database engine is closed unexpectedly because of a power outage or computer hardware problem. The database won't be marked as possibly corrupt if you use the **Close** method or if you quit your application in a usual way.

The **RepairDatabase** method also attempts to validate all system tables and all indexes. Any data that can't be repaired is discarded. If the database can't be repaired, a run-time error occurs.

When you attempt to open or compact a corrupted database, a run-time error usually occurs. In some situations, however, a corrupted database may not be detected, and no error occurs. It's a good idea to provide your users with a way to use the **RepairDatabase** method in your application if their database behaves unpredictably.

Some types of databases can become corrupted if a user ends an application without closing **Database** or **Recordset** objects and the Microsoft Jet database engine; Microsoft Windows doesn't have a chance to write data caches to disk. To avoid corrupt databases, establish procedures for closing applications and shutting down systems that ensure that all cached pages are saved to the database. In some cases, power supplies that can't be interrupted may be necessary to prevent accidental data loss during power fluctuations.

Note After repairing a database, it's also a good idea to compact it using the **CompactDatabase** method to defragment the file and to recover disk space.

Requery Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthRequeryC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthRequeryX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthRequeryA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthRequeryS"}

Updates the data in a **Recordset** object by re-executing the query on which the object is based.

Syntax

recordset.**Requery** *newquerydef*

The **Requery** method syntax has the following parts.

Part	Description
recordset	An object variable that represents an existing Microsoft Jet dynaset-, snapshot-, or forward-only–type **Recordset** object, or an ODBCDirect **Recordset** object.
newquerydef	Optional. A **Variant** that represents the **Name** property value of a **QueryDef** object (Microsoft Jet workspaces only).

Remarks

Use this method to make sure that a **Recordset** contains the most recent data. This method re-populates the current **Recordset** by using either the current query parameters or (in a Microsoft Jet workspace) the new ones supplied by the *newquerydef* argument.

In an ODBCDirect workspace, if the original query was asynchronous, then **Requery** will also execute an asynchronous query.

If you don't specify a *newquerydef* argument, the **Recordset** is re-populated based on the same query definition and parameters used to originally populate the **Recordset.** Any changes to the underlying data will be reflected during this re-population. If you didn't use a **QueryDef** to create the **Recordset**, the **Recordset** is re-created from scratch.

If you specify the original **QueryDef** in the *newquerydef* argument, then the **Recordset** is requeried using the parameters specified by the **QueryDef**. Any changes to the underlying data will be reflected during this re-population. To reflect any changes to the query parameter values in the **Recordset**, you must supply the *newquerydef* argument.

If you specify a different **QueryDef** than what was originally used to create the **Recordset**, the **Recordset** is re-created from scratch.

When you use **Requery**, the first record in the **Recordset** becomes the current record.

You can't use the **Requery** method on dynaset- or snapshot-type **Recordset** objects whose **Restartable** property is set to **False**. However, if you supply the optional *newquerydef* argument, the **Restartable** property is ignored.

If both the **BOF** and **EOF** property settings of the **Recordset** object are **True** after you use the **Requery** method, the query didn't return any records and the **Recordset** contains no data.

Seek Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthSeekC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthSeekX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthSeekA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthSeekS"}

Locates the record in an indexed table-type **Recordset** object that satisfies the specified criteria for the current index and makes that record the current record (Microsoft Jet workspaces only).

Syntax

*recordset***.Seek** *comparison*, *key1*, *key2*...*key13*

The **Seek** method syntax has the following parts.

Part	Description
recordset	An object variable that represents an existing table-type **Recordset** object that has a defined index as specified by the **Recordset** object's **Index** property.
comparison	One of the following string expressions: <, <=, =, >=, or >.
key1, *key2*...*key13*	One or more values corresponding to fields in the **Recordset** object's current index, as specified by its **Index** property setting. You can use up to 13 *key* arguments.

Remarks

You must set the current index with the **Index** property before you use **Seek**. If the index identifies a nonunique key field, **Seek** locates the first record that satisfies the criteria.

The **Seek** method searches through the specified key fields and locates the first record that satisfies the criteria specified by *comparison* and *key1*. Once found, it makes that record current and sets the **NoMatch** property to **False**. If the **Seek** method fails to locate a match, the **NoMatch** property is set to **True**, and the current record is undefined.

If *comparison* is equal (=), greater than or equal (>=), or greater than (>), **Seek** starts at the beginning of the index and searches forward.

If *comparison* is less than (<) or less than or equal (<=), **Seek** starts at the end of the index and searches backward. However, if there are duplicate index entries at the end of the index, **Seek** starts at an arbitrary entry among the duplicates and then searches backward.

You must specify values for all fields defined in the index. If you use **Seek** with a multiple-column index, and you don't specify a comparison value for every field in the index, then you cannot use the equal (=) operator in the comparison. That's because some of the criteria fields (*key2*, *key3*, and so on) will default to **Null**, which will probably not match. Therefore, the equal operator will work correctly only if you have a record which is all **Null** except the key you're looking for. It's recommended that you use the greater than or equal (>=) operator instead.

The *key1* argument must be of the same field data type as the corresponding field in the current index. For example, if the current index refers to a number field (such as Employee ID), *key1* must be numeric. Similarly, if the current index refers to a Text field (such as Last Name), *key1* must be a string.

There doesn't have to be a current record when you use **Seek**.

You can use the **Indexes** collection to enumerate the existing indexes.

To locate a record in a dynaset- or snapshot-type **Recordset** that satisfies a specific condition that is not covered by existing indexes, use the **Find** methods. To include all records, not just those that satisfy a specific condition, use the **Move** methods to move from record to record.

You can't use the **Seek** method on a linked table because you can't open linked tables as table-type **Recordset** objects. However, if you use the **OpenDatabase** method to directly open an installable ISAM (non-ODBC) database, you can use **Seek** on tables in that database.

In an ODBCDirect workspace, the **Find** and **Seek** methods are not available on any type of **Recordset** object, because executing a **Find** or **Seek** through an ODBC connection is not very efficient over the network. Instead, you should design the query (that is, using the *source* argument to the **OpenRecordset** method) with an appropriate WHERE clause that restricts the returned records to only those that meet the criteria you would otherwise use in a **Find** or **Seek**.

Synchronize Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthSynchronizeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthSynchronizeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthSynchronizeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthSynchronizeS"}

Synchronizes two replicas. (Microsoft Jet databases only).

Syntax

*database.***Synchronize** *pathname***,** *exchange*

The **Synchronize** method syntax has the following parts.

Part	Description
database	An object variable that represents a **Database** object that is a replica.
pathname	A **String** that contains the path to the target replica with which *database* will be synchronized. The .mdb file name extension is optional.
exchange	Optional. A constant indicating which direction to synchronize changes between the two databases, as specified in Settings.

Settings

You can use the following constants in the *exchange* argument. You can use one of the first three constants with or without the fourth constant.

Constant	Description
dbRepExportChanges	Sends changes from *database* to *pathname.*
dbRepImportChanges	Sends changes from *pathname* to *database.*
dbRepImpExpChanges	(Default) Sends changes from *database* to *pathname*, and vice-versa, also known as bidirectional exchange.
dbRepSyncInternet	Exchanges data between files connected by an Internet pathway.

Remarks

You use **Synchronize** to exchange data and design changes between two databases. Design changes always happen first. Both databases must be at the same design level before they can exchange data. For example, an exchange of type **dbRepExportChanges** might cause design changes at a replica even though data changes flow only from the *database* to *pathname*.

The replica identified in *pathname* must be part of the same replica set. If both replicas have the same **ReplicaID** property setting or are Design Masters for two different replica sets, the synchronization fails.

When you synchronize two replicas over the Internet, you must use the **dbRepSyncInternet** constant. In this case, you specify a Uniform Resource Locator (URL) address for the *pathname* argument instead of specifying a local area network path.

Note You can't synchronize partial replicas with other partial replicas. See the **PopulatePartial** method for more information.

Synchronization over the Internet requires the Replication Manager, which is only available in the Microsoft Office 97, Developer Edition.

Update Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthUpdateC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthUpdateX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthUpdateA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthUpdateS"}

Saves the contents of the copy buffer to an updatable **Recordset** object.

Syntax

recordset.**Update (***type*, *force* **)**

The **Update** method syntax has the following parts.

Part	Description
recordset	An object variable that represents an open, updatable **Recordset** object.
type	Optional. A constant indicating the type of update, as specified in Settings (ODBCDirect workspaces only).
force	Optional. A **Boolean** value indicating whether or not to force the changes into the database, regardless of whether the underlying data has been changed by another user since the **AddNew**, **Delete**, or **Edit** call. If **True**, the changes are forced and changes made by other users are simply overwritten. If **False** (default), changes made by another user while the update is pending will cause the update to fail for those changes that are in conflict. No error occurs, but the **BatchCollisionCount** and **BatchCollisions** properties will indicate the number of conflicts and the rows affected by conflicts, respectively (ODBCDirect workspaces only).

Settings

You can use the following values for the *type* argument. You can use the non-default values only if batch updating is enabled.

Constant	Description
dbUpdateRegular	Default. Pending changes aren’t cached and are written to disk immediately.
dbUpdateBatch	All pending changes in the update cache are written to disk.
dbUpdateCurrentRecord	Only the current record’s pending changes are written to disk.

Remarks

Use **Update** to save the current record and any changes you've made to it.

Caution Changes to the current record are lost if:

- You use the **Edit** or **AddNew** method, and then move to another record without first using **Update**.
- You use **Edit** or **AddNew**, and then use **Edit** or **AddNew** again without first using **Update**.
- You set the **Bookmark** property to another record.
- You close *recordset* without first using **Update**.
- You cancel the **Edit** operation by using **CancelUpdate**.

To edit a record, use the **Edit** method to copy the contents of the current record to the copy buffer. If

you don't use **Edit** first, an error occurs when you use **Update** or attempt to change a field's value.

In an ODBCDirect workspace, you can do batch updates, provided the cursor library supports batch updates, and the **Recordset** was opened with the optimistic batch locking option.

In a Microsoft Jet workspace, when the **Recordset** object's **LockEdits** property setting is **True** (pessimistically locked) in a multiuser environment, the record remains locked from the time **Edit** is used until the **Update** method is executed or the edit is canceled. If the **LockEdits** property setting is **False** (optimistically locked), the record is locked and compared with the pre-edited record just before it is updated in the database. If the record has changed since you used the **Edit** method, the **Update** operation fails. Microsoft Jet-connected ODBC and installable ISAM databases always use optimistic locking. To continue the **Update** operation with your changes, use the **Update** method again. To revert to the record as the other user changed it, refresh the current record by using `Move 0`.

Note To add, edit, or delete a record, there must be a unique index on the record in the underlying data source. If not, a “Permission denied” error will occur on the **AddNew**, **Delete,** or **Edit** method call in a Microsoft Jet workspace, or an “Invalid argument” error will occur on the **Update** call in an ODBCDirect workspace.

Cancel Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthCancelC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthCancelX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthCancelA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthCancelS"}

Cancels execution of a pending asynchronous method call (ODBCDirect workspaces only).

Syntax

object.**Cancel**

The **Cancel** method syntax has these parts.

Part	Description
object	A string expression that evaluates to one of the objects in the "Applies To" list.

Remarks

Use the **Cancel** method to terminate execution of an asynchronous **Execute**, **MoveLast**, **OpenConnection,** or **OpenRecordset** method call (that is, the method was invoked with the **dbRunAsync** option). **Cancel** will return a run-time error if **dbRunAsync** was not used in the method you're trying to terminate.

The following table shows what task is terminated when you use the **Cancel** method on a particular type of object.

If *object* is a	This asynchronous method is terminated
Connection	**Execute** or **OpenConnection**
QueryDef	**Execute**
Recordset	**MoveLast** or **OpenRecordset**

An error will occur if, following a **Cancel** method call, you try to reference the object that would have been created by an asynchronous **OpenConnection** or **OpenRecordset** call (that is, the **Connection** or **Recordset** object from which you called the **Cancel** method).

NextRecordset Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthNextRecordsetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthNextRecordsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthNextRecordsetA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthNextRecordsetS"}

Gets the next set of records, if any, returned by a multi-part select query in an **OpenRecordset** call, and returns a **Boolean** value indicating whether one or more additional records are pending (ODBCDirect workspaces only).

Syntax

Set *boolean* = *recordset*.**NextRecordset**

The **NextRecordset** method syntax has these parts:

Part	Description
boolean	A **Boolean** variable. **True** indicates the next set of records is available in *recordset*; **False** indicates that no more records are pending and *recordset* is now empty.
recordset	An existing **Recordset** variable to which you want to return pending records.

Remarks

In an ODBCDirect workspace, you can open a **Recordset** containing more than one select query in the ***source*** argument of **OpenRecordset**, or the **SQL** property of a select query **QueryDef** object, as in the following example.

```
SELECT LastName, FirstName FROM Authors
WHERE LastName = 'Smith';
SELECT Title, ISBN FROM Titles
WHERE Pub_ID = 9999
```

The returned **Recordset** will open with the results of the first query. To obtain the result sets of records from subsequent queries, use the **NextRecordset** method.

If more records are available (that is, there was another select query in the **OpenRecordset** call or in the **SQL** property), the records returned from the next query will be loaded into the **Recordset**, and **NextRecordset** will return **True**, indicating that the records are available. When no more records are available (that is, results of the last select query have been loaded into the **Recordset**), then **NextRecordset** will return **False**, and the **Recordset** will be empty.

You can also use the **Cancel** method to flush the contents of a **Recordset**. However, **Cancel** also flushes any additional records not yet loaded.

OpenConnection Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthOpenConnectionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthOpenConnectionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthOpenConnectionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthOpenConnectionS"}

Opens a **Connection** object on an ODBC data source (ODBCDirect workspaces only).

Syntax

Set *connection* = *workspace*.**OpenConnection (***name*, *options*, *readonly*, *connect***)**

The **OpenConnection** method syntax has these parts.

Part	Description
connection	A **Connection** object variable to which the new connection will be assigned.
workspace	Optional. A variable of a **Workspace** object data type that references the existing **Workspace** object that will contain the new connection.
name	A string expression. See the discussion under Remarks.
options	Optional. A **Variant** that sets various options for the connection, as specified in Settings. Based on this value, the ODBC driver manager prompts the user for connection information such as data source name (DSN), user name, and password.
readonly	Optional. A **Boolean** value that is **True** if the connection is to be opened for read-only access and **False** if the connection is to be opened for read/write access (default).
connect	Optional. An ODBC connect string. See the **Connect** property for the specific elements and syntax of this string. A prepended "ODBC;" is required. If *connect* is omitted, the UID and/or PWD will be taken from the **UserName** and **Password** properties of the **Workspace**.

Settings

The *options* argument determines if and when to prompt the user to establish the connection, and whether or not to open the connection asynchronously. You can use one of the following constants.

Constant	Description
dbDriverNoPrompt	The ODBC Driver Manager uses the connection string provided in *dbname* and *connect*. If you don't provide sufficient information, a run-time error occurs.

dbDriverPrompt	The ODBC Driver Manager displays the **ODBC Data Sources** dialog box, which displays any relevant information supplied in *dbname* or *connect*. The connection string is made up of the DSN that the user selects via the dialog boxes, or, if the user doesn't specify a DSN, the default DSN is used.
dbDriverComplete	Default. If the *connect* argument includes all the necessary information to complete a connection, the ODBC Driver Manager uses the string in *connect*. Otherwise it behaves as it does when you specify **dbDriverPrompt**.
dbDriverCompleteRequired	This option behaves like **dbDriverComplete** except the ODBC driver disables the prompts for any information not required to complete the connection.
dbRunAsync	Execute the method asynchronously. This constant may be used with any of the other *options* constants.

Remarks

Use the **OpenConnection** method to establish a connection to an ODBC data source from an ODBCDirect workspace. The **OpenConnection** method is similar but not equivalent to **OpenDatabase**. The main difference is that **OpenConnection** is available only in an ODBCDirect workspace.

If you specify a registered ODBC data source name (DSN) in the *connect* argument, then the *name* argument can be any valid string, and will also provide the **Name** property for the **Connection** object. If a valid DSN is not included in the *connect* argument, then *name* must refer to a valid ODBC DSN, which will also be the **Name** property. If neither *name* nor *connect* contains a valid DSN, the ODBC driver manager can be set (via the *options* argument) to prompt the user for the required connection information. The DSN supplied through the prompt then provides the **Name** property.

OpenConnection returns a **Connection** object which contains information about the connection. The **Connection** object is similar to a **Database** object. The principal difference is that a **Database** object usually represents a database, although it can be used to represent a connection to an ODBC data source from a Microsoft Jet workspace.

SetOption Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthSetOptionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthSetOptionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthSetOptionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthSetOptionS"}

Temporarily overrides values for the Microsoft Jet database engine keys in the Windows Registry (Microsoft Jet workspaces only).

Syntax

DBEngine.SetOption *parameter*, *newvalue*

The **SetOption** method syntax has these parts.

Part	Description
parameter	A **Long** constant as described in Settings.
newvalue	A **Variant** value that you want to set *parameter* to.

Settings

Each constant refers to the corresponding registry key in the path Jet\3.5\Engines\Jet 3.5\ (that is, **dbSharedAsyncDelay** corresponds to the key Jet\3.5\Engines\Jet 3.5\SharedAsyncDelay, and so on.).

Constant	Description
dbPageTimeout	The PageTimeout key
dbSharedAsyncDelay	The SharedAsyncDelay key
dbExclusiveAsyncDelay	The ExclusiveAsyncDelay key
dbLockRetry	The LockRetry key
dbUserCommitSync	The UserCommitSync key
dbImplicitCommitSync	The ImplicitCommitSync key
dbMaxBufferSize	The MaxBufferSize key
dbMaxLocksPerFile	The MaxLocksPerFile key
dbLockDelay	The LockDelay key
dbRecycleLVs	The RecycleLVs key
dbFlushTransactionTimeout	The FlushTransactionTimeout key

Remarks

Use the **SetOption** method to override registry values at run-time. New values established with the **SetOption** method remain in effect until changed again by another **SetOption** call, or until the **DBEngine** object is closed.

For further details on what the registry keys do, and appropriate values to set them to, see Initializing the Microsoft Jet 3.5 Database Engine.

PopulatePartial Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damthPopulatePartialC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damthPopulatePartialX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"damthPopulatePartialA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damthPopulatePartialS"}

Synchronizes any changes in a partial replica with the full replica, clears all records in the partial replica, and then repopulates the partial replica based on the current replica filters. (Microsoft Jet databases only.)

Syntax

database.**PopulatePartial** *dbname*

The **PopulatePartial** method syntax has the following parts.

Part	Description
database	An object variable that references the partial replica **Database** object that you want to populate.
dbname	A **string** specifying the path and name of the full replica from which to populate records.

Remarks

When you synchronize a partial replica with a full replica, it is possible to create "orphaned" records in the partial replica. For example, suppose you have a Customers table with its **ReplicaFilter** set to `"Region = 'CA'"`. If a user changes a customer's region from CA to NY in the partial replica, and then a synchronization occurs via the **Synchronize** method, the change is propagated to the full replica but the record containing NY in the partial replica is orphaned because it now doesn't meet the replica filter criteria.

To solve the problem of orphaned records, you can use the **PopulatePartial** method. The **PopulatePartial** method is similar to the **Synchronize** method, but it synchronizes any changes with the full replica, removes all records in the partial replica, and then repopulates the partial replica based on the current replica filters. Even if your replica filters have not changed, **PopulatePartial** will always clear all records in the partial replica and repopulate it based on the current filters.

Generally, you should use the **PopulatePartial** method when you create a partial replica and whenever you change your replica filters. If your application changes replica filters, you should follow these steps:

1. Synchronize your full replica with the partial replica in which the filters are being changed.
2. Use the **ReplicaFilter** and **PartialReplica** properties to make the desired changes to the replica filter.
3. Call the **PopulatePartial** method to remove all records from the partial replica and transfer all records from the full replica that meet the new replica filter criteria.

If a replica filter has changed, and the **Synchronize** method is invoked without first invoking **PopulatePartial**, a trappable error occurs.

The **PopulatePartial** method can only be invoked on a partial replica that has been opened for exclusive access. Furthermore, you can't call the **PopulatePartial** method from code running within the partial replica itself. Instead, open the partial replica exclusively from the full replica or another database, then call **PopulatePartial**.

Note Although **PopulatePartial** performs a one-way synchronization before clearing and repopulating the partial replica, it is still a good idea to call **Synchronize** before calling **PopulatePartial**. This is because if the call to **Synchronize** fails, a trappable error occurs. You can

use this error to decide whether or not to proceed with the **PopulatePartial** method (which removes all records in the partial replica). If **PopulatePartial** is called by itself and an error occurs while records are being synchronized, records in the partial replica will still be cleared, which may not be the desired result.

AbsolutePosition Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproAbsolutePositionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproAbsolutePositionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproAbsolutePositionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproAbsolutePositionS"}

Sets or returns the relative record number of a **Recordset** object's current record.

Settings and Return Values

The setting or return value is a **Long** integer from 0 to one less than the number of records in the **Recordset** object. It corresponds to the ordinal position of the current record in the **Recordset** object specified by the object.

Remarks

You can use the **AbsolutePosition** property to position the current record pointer to a specific record based on its ordinal position in a dynaset- or snapshot-type **Recordset** object. You can also determine the current record number by checking the **AbsolutePosition** property setting.

Because the **AbsolutePosition** property value is zero-based (that is, a setting of 0 refers to the first record in the **Recordset** object), you cannot set it to a value greater than or equal to the number of populated records; doing so causes a trappable error. You can determine the number of populated records in the **Recordset** object by checking the **RecordCount** property setting. The maximum allowable setting for the **AbsolutePosition** property is the value of the **RecordCount** property minus 1.

If there is no current record, as when there are no records in the **Recordset** object, **AbsolutePosition** returns –1. If the current record is deleted, the **AbsolutePosition** property value isn't defined, and a trappable error occurs if it's referenced. New records are added to the end of the sequence.

You shouldn't use this property as a surrogate record number. Bookmarks are still the recommended way of retaining and returning to a given position and are the only way to position the current record across all types of **Recordset** objects. In particular, the position of a record changes when one or more records preceding it are deleted. There is also no assurance that a record will have the same absolute position if the **Recordset** object is re-created again because the order of individual records within a **Recordset** object isn't guaranteed unless it's created with an SQL statement by using an ORDER BY clause.

Notes

- Setting the **AbsolutePosition** property to a value greater than zero on a newly opened but unpopulated **Recordset** object causes a trappable error. Populate the **Recordset** object first with the **MoveLast** method.
- The **AbsolutePosition** property isn't available on forward-only–type **Recordset** objects, or on **Recordset** objects opened from pass-through queries against Microsoft Jet-connected ODBC databases.

AllowZeroLength Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproAllowZeroLengthC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproAllowZeroLengthX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproAllowZeroLengthA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproAllowZeroLengthS"}

Sets or returns a value that indicates whether a zero-length string ("") is a valid setting for the **Value** property of the **Field** object with a **Text** or **Memo** data type.

Settings and Return Values

The setting or return value is a **Boolean** data type that indicates if a value is valid. The value is **True** if the **Field** object accepts a zero-length string as its **Value** property; the default value is **False**.

Remarks

For an object not yet appended to the **Fields** collection, this property is read/write.

Once appended to a **Fields** collection, the availability of the **AllowZeroLength** property depends on the object that contains the **Fields** collection, as shown in the following table.

If the Fields collection belongs to an	Then AllowZeroLength is
Index object	Not supported
QueryDef object	Read-only
Recordset object	Read-only
Relation object	Not supported
TableDef object	Read/write

You can use this property along with the **Required**, **ValidateOnSet**, or **ValidationRule** property to validate a value in a field.

AllPermissions Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproAllPermissionsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproAllPermissionsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproAllPermissionsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproAllPermissionsS"}

Returns all the permissions that apply to the current **UserName** property of the **Container** or **Document** object, including permissions that are specific to the user as well as the permissions a user inherits from memberships in groups (Microsoft Jet workspaces only).

Return Values

For any **Container** or **Document** object, the return value is a **Long** value or constant(s) that may include the following.

Constant	Description
dbSecReadDef	The user can read the table definition, including column and index information.
dbSecWriteDef	The user can modify or delete the table definition, including column and index information.
dbSecRetrieveData	The user can retrieve data from the **Document** object.
dbSecInsertData	The user can add records.
dbSecReplaceData	The user can modify records.
dbSecDeleteData	The user can delete records.

In addition, the Databases container or any **Document** object in a **Documents** collection may include the following.

Constant	Description
dbSecDeleteData	The user can delete records.
dbSecDBAdmin	The user can replicate the database and change the database password.
dbSecDBCreate	The user can create new databases. This setting is valid only on the Databases container in the workgroup information file (System.mdw).
dbSecDBExclusive	The user has exclusive access to the database.
dbSecDBOpen	The user can open the database.

Remarks

This property contrasts with the **Permissions** property, which returns only the permissions that are specific to the user and doesn't include any permissions that the user may also have as a member of groups. If the current value of the **UserName** property is a group, then the **AllPermissions** property returns the same values as the **Permissions** property.

Attributes Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproAttributesC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproAttributesX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproAttributesA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproAttributesS"}

Sets or returns a value that indicates one or more characteristics of a **Field**, **Relation**, or **TableDef** object.

Settings and Return Values

The setting or return value is **Long** data type, and the default value is 0.

For a **Field** object, the value specifies characteristics of the field represented by the **Field** object and can be a combination of these constants.

Constant	Description
dbAutoIncrField	The field value for new records is automatically incremented to a unique **Long** integer that can't be changed (in a Microsoft Jet workspace, supported only for Microsoft Jet database(.mdb) tables).
dbDescending	The field is sorted in descending (Z to A or 100 to 0) order; this option applies only to a **Field** object in a **Fields** collection of an **Index** object. If you omit this constant, the field is sorted in ascending (A to Z or 0 to 100) order. This is the default value for **Index** and **TableDef** fields (Microsoft Jet workspaces only).
dbFixedField	The field size is fixed (default for Numeric fields).
dbHyperlinkField	The field contains hyperlink information (Memo fields only).
dbSystemField	The field stores replication information for replicas; you can't delete this type of field (Microsoft Jet workspaces only).
dbUpdatableField	The field value can be changed.
dbVariableField	The field size is variable (Text fields only).

For a **Relation** object, the value specifies characteristics of the relationship represented by the **Relation** object and can be a combination of these constants.

Constant	Description
dbRelationUnique	The relationship is one-to-one.
dbRelationDontEnforce	The relationship isn't enforced (no referential integrity).
dbRelationInherited	The relationship exists in a non-current database that contains the two linked tables.
dbRelationUpdateCascade	Updates will cascade.
dbRelationDeleteCascade	Deletions will cascade.

Note If you set the **Relation** object's **Attributes** property to activate cascading operations, the Microsoft Jet database engine automatically updates or deletes records in one or more other tables when changes occur in related primary tables.

For example, suppose you establish a cascading delete relationship between a Customers table and an Orders table. When you delete records from the Customers table, records in the Orders table related to that customer are also deleted. In addition, if you establish cascading delete relationships between the Orders table and other tables, records from those tables are automatically deleted when you delete records from the Customers table.

For a **TableDef** object, the value specifies characteristics of the table represented by the **TableDef** object and can be a combination of these **Long** constants.

Constant	Description
dbAttachExclusive	For databases that use the Microsoft Jet database engine, the table is a linked table opened for exclusive use. You can set this constant on an appended **TableDef** object for a local table, but not on a remote table.
dbAttachSavePWD	For databases that use the Microsoft Jet database engine, the user ID and password for the remotely linked table are saved with the connection information. You can set this constant on an appended **TableDef** object for a remote table, but not on a local table.
dbSystemObject	The table is a system table provided by the Microsoft Jet database engine. You can set this constant on an appended **TableDef** object.
dbHiddenObject	The table is a hidden table provided by the Microsoft Jet database engine. You can set this constant on an appended **TableDef** object.
dbAttachedTable	The table is a linked table from a non-ODBC data source such as a Microsoft Jet or Paradox database (read-only).
dbAttachedODBC	The table is a linked table from an ODBC data source, such as Microsoft SQL Server (read-only).

Remarks

For an object not yet appended to a collection, this property is read/write.

For an appended **Field** object, the availability of the **Attributes** property depends on the object that contains the **Fields** collection.

If the Field object belongs to an	Then Attributes is
Index object	Read/write until the **TableDef** object that the **Index** object is appended to is appended to a **Database** object; then the property is read-only.

QueryDef object	Read-only
Recordset object	Read-only
Relation object	Not supported
TableDef object	Read/write

For an appended **Relation** object, the **Attributes** property setting is read-only.

For an appended **TableDef** object, the property is read/write, although you can't set all of the constants if the object is appended, as noted in Settings and Return Values.

When you set multiple attributes, you can combine them by summing the appropriate constants. Any invalid values are ignored without producing an error.

BOF, EOF Properties

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproBOFC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproBOFX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproBOFA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproBOFS"}

- **BOF** – returns a value that indicates whether the current record position is before the first record in a **Recordset** object.
- **EOF** – returns a value that indicates whether the current record position is after the last record in a **Recordset** object.

Return Values

The return values for the **BOF** and **EOF** properties are **Boolean** values.

The **BOF** property returns **True** if the current record position is before the first record, and **False** if the current record position is on or after the first record.

The **EOF** property returns **True** if the current record position is after the last record, and **False** if the current record position is on or before the last record.

Remarks

You can use the **BOF** and **EOF** properties to determine whether a **Recordset** object contains records or whether you've gone beyond the limits of a **Recordset** object when you move from record to record.

The location of the current record pointer determines the **BOF** and **EOF** return values.

If either the **BOF** or **EOF** property is **True**, there is no current record.

If you open a **Recordset** object containing no records, the **BOF** and **EOF** properties are set to **True**, and the **Recordset** object's **RecordCount** property setting is 0. When you open a **Recordset** object that contains at least one record, the first record is the current record and the **BOF** and **EOF** properties are **False**; they remain **False** until you move beyond the beginning or end of the **Recordset** object by using the **MovePrevious** or **MoveNext** method, respectively. When you move beyond the beginning or end of the **Recordset**, there is no current record or no record exists.

If you delete the last remaining record in the **Recordset** object, the **BOF** and **EOF** properties may remain **False** until you attempt to reposition the current record.

If you use the **MoveLast** method on a **Recordset** object containing records, the last record becomes the current record; if you then use the **MoveNext** method, the current record becomes invalid and the **EOF** property is set to **True**. Conversely, if you use the **MoveFirst** method on a **Recordset** object containing records, the first record becomes the current record; if you then use the **MovePrevious** method, there is no current record and the **BOF** property is set to **True**.

Typically, when you work with all the records in a **Recordset** object, your code will loop through the records by using the **MoveNext** method until the **EOF** property is set to **True**.

If you use the **MoveNext** method while the **EOF** property is set to **True** or the **MovePrevious** method while the **BOF** property is set to **True**, an error occurs.

This table shows which Move methods are allowed with different combinations of the **BOF** and **EOF** properties.

	MoveFirst, MoveLast	**MovePrevious, Move < 0**	**Move 0**	**MoveNext, Move > 0**
BOF=True, EOF=False	Allowed	Error	Error	Allowed

BOF=False, EOF=True	Allowed	Allowed	Error	Error
Both **True**	Error	Error	Error	Error
Both **False**	Allowed	Allowed	Allowed	Allowed

Allowing a Move method doesn't mean that the method will successfully locate a record. It merely indicates that an attempt to perform the specified Move method is allowed and won't generate an error. The state of the **BOF** and **EOF** properties may change as a result of the attempted Move.

An **OpenRecordset** method internally invokes a **MoveFirst** method. Therefore, using an **OpenRecordset** method on an empty set of records sets the **BOF** and **EOF** properties to **True**. (See the following table for the behavior of a failed **MoveFirst** method.)

All Move methods that successfully locate a record will set both **BOF** and **EOF** to **False**.

In a Microsoft Jet workspace, if you add a record to an empty **Recordset**, **BOF** will become **False**, but **EOF** will remain **True**, indicating that the current position is at the end of **Recordset**. In an ODBCDirect workspace, both **BOF** and **EOF** will become **False**, indicating that the current position is on the new record.

Any **Delete** method, even if it removes the only remaining record from a **Recordset**, won't change the setting of the **BOF** or **EOF** property.

The following table shows how Move methods that don't locate a record affect the **BOF** and **EOF** property settings.

	BOF	EOF
MoveFirst, **MoveLast**	**True**	**True**
Move 0	No change	No change
MovePrevious, **Move** < 0	**True**	No change
MoveNext, **Move** > 0	No change	**True**

Bookmark Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproBookmarkC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproBookmarkX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproBookmarkA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproBookmarkS"}

Sets or returns a bookmark that uniquely identifies the current record in a **Recordset** object.

Settings and Return Values

The setting or return value is a string expression or variant expression that evaluates to a valid bookmark. The data type is a **Variant** array of **Byte** data.

Remarks

For a **Recordset** object based entirely on Microsoft Jet tables, the value of the **Bookmarkable** property is **True**, and you can use the **Bookmark** property with that **Recordset**. Other database products may not support bookmarks, however. For example, you can't use bookmarks in any **Recordset** object based on a linked Paradox table that has no primary key.

When you create or open a **Recordset** object, each of its records already has a unique bookmark. You can save the bookmark for the current record by assigning the value of the **Bookmark** property to a variable. To quickly return to that record at any time after moving to a different record, set the **Recordset** object's **Bookmark** property to the value of that variable.

There is no limit to the number of bookmarks you can establish. To create a bookmark for a record other than the current record, move to the desired record and assign the value of the **Bookmark**

property to a **String** variable that identifies the record.

To make sure the **Recordset** object supports bookmarks, check the value of its **Bookmarkable** property before you use the **Bookmark** property. If the **Bookmarkable** property is **False**, the **Recordset** object doesn't support bookmarks, and using the **Bookmark** property results in a trappable error.

If you use the **Clone** method to create a copy of a **Recordset** object, the **Bookmark** property settings for the original and the duplicate **Recordset** objects are identical and can be used interchangeably. However, you can't use bookmarks from different **Recordset** objects interchangeably, even if they were created by using the same object or the same SQL statement.

If you set the **Bookmark** property to a value that represents a deleted record, a trappable error occurs.

The value of the **Bookmark** property isn't the same as a record number.

Bookmarkable Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproBookmarkableC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproBookmarkableX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproBookmarkableA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproBookmarkableS"}

Returns a value that indicates whether a **Recordset** object supports bookmarks, which you can set by using the **Bookmark** property.

Return Values

The return value is a **Boolean** data type that returns **True** if the object supports bookmarks.

Remarks

Check the **Bookmarkable** property setting of a **Recordset** object before you attempt to set or check the **Bookmark** property.

For **Recordset** objects based entirely on Microsoft Jet tables, the value of the **Bookmarkable** property is **True**, and you can use bookmarks. Other database products may not support bookmarks, however. For example, you can't use bookmarks in any **Recordset** object based on a linked Paradox table that has no primary key.

CacheSize Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproCacheSizeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproCacheSizeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproCacheSizeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproCacheSizeS"}

Sets or returns the number of records retrieved from an ODBC data source that will be cached locally.

Settings and Return Values

The setting or return value is a **Long** value and must be between 5 and 1200, but not greater than available memory will allow. A typical value is 100. A setting of 0 turns off caching.

Remarks

Data caching improves performance if you use **Recordset** objects to retrieve data from a remote server. A cache is a space in local memory that holds the data most recently retrieved from the server; this is useful if users request the data again while the application is running. When users request data, the Microsoft Jet database engine checks the cache for the requested data first rather than retrieving it from the server, which takes more time. The cache only saves data that comes from an ODBC data source.

Any Microsoft Jet-connected ODBC data source, such as a linked table, can have a local cache. To create the cache, open a **Recordset** object from the remote data source, set the **CacheSize** and **CacheStart** properties, and then use the **FillCache** method, or step through the records by using the Move methods.

An ODBCDirect workspace can use a local cache. To create the cache, set the **CacheSize** property on a **QueryDef** object. On a **Relation** object, **CacheSize** is read-only and depends on the value of the **QueryDef** object's **CacheSize** property. You can't use the **CacheStart** property on **FillCache** method in an ODBCDirect workspace.

You can base the **CacheSize** property setting on the number of records your application can handle at one time. For example, if you're using a **Recordset** object as the source of the data to be displayed on screen, you could set its **CacheSize** property to 20 to display 20 records at one time.

The Microsoft Jet database engine requests records within the cache range from the cache, and it requests records outside the cache range from the server.

Records retrieved from the cache don't reflect concurrent changes that other users made to the source data.

To force an update of all the cached data, set the **CacheSize** property of the **Recordset** object to 0, re-set it to the size of the cache you originally requested, and then use the **FillCache** method.

Clustered Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproClusteredC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproClusteredX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproClusteredA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproClusteredS"}

Sets or returns a value that indicates whether an **Index** object represents a clustered index for a table (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **Boolean** data type that is **True** if the **Index** object represents a clustered index.

Remarks

Some IISAM desktop database formats use clustered indexes. A clustered index consists of one or more nonkey fields that, taken together, arrange all records in a table in a predefined order. A clustered index provides efficient access to records in a table in which the index values may not be unique.

The **Clustered** property is read/write for a new **Index** object not yet appended to a collection and read-only for an existing **Index** object in an **Indexes** collection.

Notes

- Microsoft Jet databases ignore the **Clustered** property because the Microsoft Jet database engine doesn't support clustered indexes.
- For ODBC data sources the **Clustered** property always returns **False**; it does not detect whether or not the ODBC data source has a clustered index.

CollatingOrder Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproCollatingOrderC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproCollatingOrderX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproCollatingOrderA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproCollatingOrderS"}

Returns a value that specifies the sequence of the sort order in text for string comparison or sorting (Microsoft Jet workspaces only).

Return Values

The return value is a **Long** value or constant that can be one of the following values.

Constant	Sort order
dbSortGeneral	General (English, French, German, Portuguese, Italian, and Modern Spanish)
dbSortArabic	Arabic
dbSortChineseSimplified	Simplified Chinese
dbSortChineseTraditional	Traditional Chinese
dbSortCyrillic	Russian
dbSortCzech	Czech
dbSortDutch	Dutch
dbSortGreek	Greek
dbSortHebrew	Hebrew
dbSortHungarian	Hungarian
dbSortIcelandic	Icelandic
dbSortJapanese	Japanese
dbSortKorean	Korean
dbSortNeutral	Neutral
dbSortNorwDan	Norwegian or Danish
dbSortPDXIntl	Paradox International
dbSortPDXNor	Paradox Norwegian or Danish
dbSortPDXSwe	Paradox Swedish or Finnish
dbSortPolish	Polish
dbSortSlovenian	Slovenian
dbSortSpanish	Spanish
dbSortSwedFin	Swedish or Finnish
dbSortThai	Thai
dbSortTurkish	Turkish
dbSortUndefined	Undefined or unknown

Remarks

The availability of the **CollatingOrder** property depends on the object that contains the **Fields** collection, as shown in the following table.

If the Fields collection belongs to an	Then CollatingOrder is
Index object	Not supported
QueryDef object	Read-only
Recordset object	Read-only

Relation object	Not supported
TableDef object	Read-only

The **CollatingOrder** property setting corresponds to the *locale* argument of the **CreateDatabase** method when the database was created or the **CompactDatabase** method when the database was most recently compacted.

Check the **CollatingOrder** property setting of a **Database** or **Field** object to determine the string comparison method for the database or field. You can set the **CollatingOrder** property of a new, unappended **Field** object if you want the setting of the **Field** object to differ from that of the **Database** object that contains it.

The **CollatingOrder** and **Attributes** property settings of a **Field** object in a **Fields** collection of an **Index** object together determine the sequence and direction of the sort order in an index. However, you can't set a collating order for an individual index—you can only set it for an entire table.

ConflictTable Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproConflictTableC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproConflictTableX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproConflictTableA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproConflictTableS"}

Returns the name of a conflict table containing the database records that conflicted during the synchronization of two replicas (Microsoft Jet workspaces only).

Return Values

The return value is a **String** data type that is a zero-length string if there is no conflict table or the database isn't a replica.

Remarks

If two users at two separate replicas each make a change to the same record in the database, the changes made by one user will fail to be applied to the other replica. Consequently, the user with the failed change must resolve the conflicts.

Conflicts occur at the record level, not between fields. For example, if one user changes the Address field and another updates the Phone field in the same record, then one change is rejected. Because conflicts occur at the record level, the rejection occurs even though the successful change and the rejected change are unlikely to result in a true conflict of information.

The synchronization mechanism handles the record conflicts by creating conflict tables, which contain the information that would have been placed in the table, if the change had been successful. You can examine these conflict tables and work through them row by row, fixing whatever is appropriate.

All conflict tables are named *table*_conflict, where *table* is the original name of the table, truncated to the maximum table name length.

Connect Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproConnectC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproConnectX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproConnectA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproConnectS"}

Sets or returns a value that provides information about the source of an open connection, an open database, a database used in a pass-through query, or a linked table. For **Database** objects, new **Connection** objects, linked tables, and **TableDef** objects not yet appended to a collection, this property setting is read/write. For **QueryDef** objects and base tables, this property is read-only.

Syntax

object.**Connect** = *databasetype*;*parameters*;

The **Connect** property syntax has these parts.

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
databasetype	Optional. A **String** that specifies a database type. For Microsoft Jet databases, exclude this argument; if you specify *parameters*, use a semicolon (;) as a placeholder.
parameters	Optional. A **String** that specifies additional parameters to pass to ODBC or installable ISAM drivers. Use semicolons to separate parameters.

Settings

The **Connect** property setting is a **String** composed of a database type specifier and zero or more parameters separated by semicolons. The **Connect** property passes additional information to ODBC and certain ISAM drivers as needed.

To perform an SQL pass-through query on a table linked to your Microsoft Jet database (.mdb) file, you must first set the **Connect** property of the linked table's database to a valid ODBC connection string.

For a **TableDef** object that represents a linked table, the **Connect** property setting consists of one or two parts (a database type specifier and a path to the database), each of which ends with a semicolon.

The path as shown in the following table is the full path for the directory containing the database files and must be preceded by the identifier `DATABASE=`. In some cases (as with Microsoft Excel and Microsoft Jet databases), you should include a specific file name in the database path argument.

The following table shows possible database types and their corresponding database specifiers and paths for the **Connect** property setting. You can also specify “FTP://*path*/*etc.*” or “HTTP://*path*/*etc.*” For the path. In an ODBCDirect workspace, only the "ODBC" specifier can be used.

Database type	Specifier	Example
Microsoft Jet Database	[*database*];	*drive*:*path**filename.mdb*
dBASE III	dBASE III;	*drive*:*path*
dBASE IV	dBASE IV;	*drive*:*path*
dBASE 5	dBASE 5.0;	*drive:**path*
Paradox 3.x	Paradox 3.x;	*drive*:*path*
Paradox 4.x	Paradox 4.x;	*drive*:*path*

Paradox 5.x	Paradox 5.x;	*drive:\path*
FoxPro 2.0	FoxPro 2.0;	*drive*:*path*
FoxPro 2.5	FoxPro 2.5;	*drive*:*path*
FoxPro 2.6	FoxPro 2.6;	*drive:\path*
Excel 3.0	Excel 3.0;	*drive:\path\filename.xls*
Excel 4.0	Excel 4.0;	*drive:\path\filename.xls*
Excel 5.0 or Excel 95	Excel 5.0;	*drive:\path\filename.xls*
Excel 97	Excel 97;	*drive:\path\filename.xls*
HTML Import	HTML Import;	*drive:\path\filename*
HTML Export	HTML Export;	*drive:\path*
Text	Text;	*drive:\path*
ODBC	ODBC; DATABASE=*database*; UID=*user*; PWD=*password*; DSN= *datasourcename*; [LOGINTIMEOUT=*seconds*;]	None
Exchange	Exchange; MAPILEVEL=*folderpath*; [TABLETYPE={ 0 | 1 }]; [PROFILE=*profile*;] [PWD=*password*;] [DATABASE=*database*;]	*drive*:*path**filename.mdb*

Remarks

If the specifier is only `"ODBC;"`, the ODBC driver displays a dialog box listing all registered ODBC data source names so that the user can select a database.

If a password is required but not provided in the **Connect** property setting, a login dialog box is displayed the first time a table is accessed by the ODBC driver and again if the connection is closed and reopened.

For data in Microsoft Exchange, the required MAPILEVEL key should be set to a fully-resolved folder path (for example, "Mailbox - Pat SmithIAlpha/Today"). The path does not include the name of the folder that will be opened as a table; that folder's name should instead be specified as the *name* argument to the **CreateTable** method. The TABLETYPE key should be set to "0" to open a folder (default) or "1" to open an address book. The PROFILE key defaults to the profile currently in use.

For base tables in a Micorosoft Jet database (.mdb), the **Connect** property setting is a zero-length string ("").

You can set the **Connect** property for a **Database** object by providing a *source* argument to the **OpenDatabase** method. You can check the setting to determine the type, path, user ID, password, or ODBC data source of the database.

On a **QueryDef** object in a Microsoft Jet workspace, you can use the **Connect** property with the **ReturnsRecords** property to create an ODBC SQL pass-through query. The *databasetype* of the connection string is `"ODBC;"`, and the remainder of the string contains information specific to the ODBC driver used to access the remote data. For more information, see the documentation for the specific driver.

Notes

- You must set the **Connect** property before you set the **ReturnsRecords** property.

- You must have access permissions to the computer that contains the database server you're trying to access.

Container Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproContainerC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproContainerX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproContainerA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproContainerS"}

Returns the name of the **Container** object to which a **Document** object belongs (Microsoft Jet workspaces only).

Return Values

The return value is a **String** data type.

Count Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproCountC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproCountX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproCountA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproCountS"}

Returns the number of objects in a collection.

Return Value

The return value is an **Integer** data type.

Remarks

Because members of a collection begin with 0, you should always code loops starting with the 0 member and ending with the value of the **Count** property minus 1. If you want to loop through the members of a collection without checking the **Count** property, you can use a **For Each...Next** command.

The **Count** property setting is never **Null**. If its value is 0, there are no objects in the collection.

DataUpdatable Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDataUpdatableC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDataUpdatableX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDataUpdatableA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDataUpdatableS"}

Returns a value that indicates whether the data in the field represented by a **Field** object is updatable.

Return Values

The return value is a **Boolean** data type that returns **True** if the data in the field is updatable.

Remarks

Use this property to determine whether you can change the **Value** property setting of a **Field** object. This property is always **False** on a **Field** object whose **Attributes** property is **dbAutoIncrField**.

You can use the **DataUpdatable** property on **Field** objects that are appended to the **Fields** collection of **QueryDef**, **Recordset**, and **Relation** objects, but not the **Fields** collection of **Index** or **TableDef** objects.

DateCreated, LastUpdated Properties

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDateCreatedC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDateCreatedX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDateCreatedA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDateCreatedS"}

- **DateCreated** – returns the date and time that an object was created, or the date and time a base table was created if the object is a table-type **Recordset** object (Microsoft Jet workspaces only).
- **LastUpdated** – returns the date and time of the most recent change made to an object, or to a base table if the object is a table-type **Recordset** object (Microsoft Jet workspaces only).

Return Values

The return value is a **Variant** (**Date/Time** subtype).

Remarks

For table-type **Recordset** objects, the date and time settings are derived from the computer on which the base table was created or last updated. For other objects, **DateCreated** and **LastUpdated** return the date and time that the object was created or last updated. In a multiuser environment, users should get these settings directly from the file server to avoid discrepancies in the **DateCreated** and **LastUpdated** property settings.

DefaultUser, DefaultPassword Properties

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDefaultUserC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDefaultUserX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDefaultUserA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDefaultUserS"}

- **DefaultUser** – sets the user name used to create the default **Workspace** when it is initialized.
- **DefaultPassword** – sets the password used to create the default **Workspace** when it is initialized.

Settings

The setting for **DefaultUser** is a **String** data type. It can be 1–20 characters long in Microsoft Jet workspaces and any length in ODBCDirect workspaces, and it can include alphabetic characters, accented characters, numbers, spaces, and symbols except for: " (quotation marks), / (forward slash), \ (backslash), [ ] (brackets), : (colon), | (pipe), < (less-than sign), > (greater-than sign), + (plus sign), = (equal sign), ; (semicolon), , (comma), ? (question mark), * (asterisk), leading spaces, and control characters (ASCII 00 to ASCII 31).

The setting for **DefaultPassword** is a **String** data type that can be up to 14 characters long in Microsoft Jet databases and any length in ODBCDirect connections. It can contain any character except ASCII 0.

By default, the **DefaultUser** property is set to "admin" and the **DefaultPassword** property is set to a zero-length string ("").

Remarks

User names aren't usually case-sensitive; however, if you're re-creating a user account that was deleted or created in a different workgroup, the user name must be an exact case-sensitive match of the original name. Passwords are case-sensitive.

Typically, you use the **CreateWorkspace** method to create a **Workspace** object with a given user name and password. However, for backward compatibility with earlier versions and for convenience when you don't implement a secured database, the Microsoft Jet database engine automatically creates a default **Workspace** object when needed if one isn't already open. In this case, the **DefaultUser** and **DefaultPassword** property values define the user and password for the default **Workspace** object.

For this property to take effect, you should set it before calling any DAO methods.

DefaultValue Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDefaultValueC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDefaultValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDefaultValueA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDefaultValueS"}

Sets or returns the default value of a **Field** object. For a **Field** object not yet appended to the **Fields** collection, this property is read/write (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **String** data type that can contain a maximum of 255 characters. It can be either text or an expression. If the property setting is an expression, it can't contain user-defined functions, Microsoft Jet database engine SQL aggregate functions, or references to queries, forms, or other **Field** objects.

Note You can also set the **DefaultValue** property of a **Field** object on a **TableDef** object to a special value called "GenUniqueID( )". This causes a random number to be assigned to this field whenever a new record is added or created, thereby giving each record a unique identifier. The field's **Type** property must be **Long**.

Remarks

The availability of the **DefaultValue** property depends on the object that contains the **Fields** collection, as shown in the following table.

If the Fields collection belongs to an	Then DefaultValue is
Index object	Not supported
QueryDef object	Read-only
Recordset object	Read-only
Relation object	Not supported
TableDef object	Read/write

When a new record is created, the **DefaultValue** property setting is automatically entered as the value for the field. You can change the field value by setting its **Value** property.

The **DefaultValue** property doesn't apply to AutoNumber and Long Binary fields.

Description Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDescriptionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDescriptionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDescriptionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDescriptionS"}

Returns a descriptive string associated with an error.

Return Values

The return value is a **String** data type that describes the error.

Remarks

The **Description** property comprises a short description of the error. Use this property to alert the user about an error that you cannot or do not want to handle.

DistinctCount Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDistinctCountC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDistinctCountX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDistinctCountA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDistinctCountS"}

Returns a value that indicates the number of unique values for the **Index** object that are included in the associated table (Microsoft Jet workspaces only).

Return Values

The return value is a **Long** data type.

Remarks

Check the **DistinctCount** property to determine the number of unique values, or keys, in an index. Any key is counted only once, even though there may be multiple occurrences of that value if the index permits duplicate values. This information is useful in applications that attempt to optimize data access by evaluating index information. The number of unique values is also known as the *cardinality* of an **Index** object.

The **DistinctCount** property won't always reflect the actual number of keys at a particular time. For example, a change caused by a rolled back transaction won't be reflected immediately in the **DistinctCount** property. The **DistinctCount** property value also may not reflect the deletion of records with unique keys. The number will be accurate immediately after you use the **CreateIndex** method.

EditMode Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproEditModeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproEditModeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproEditModeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproEditModeS"}

Returns a value that indicates the state of editing for the current record.

Return Values

The return value is a **Long** that indicates the state of editing, as listed in the following table.

Constant	Description
dbEditNone	No editing operation is in progress.
DbEditInProgress	The **Edit** method has been invoked, and the current record is in the copy buffer.
dbEditAdd	The **AddNew** method has been invoked, and the current record in the copy buffer is a new record that hasn't been saved in the database.

Remarks

The **EditMode** property is useful when an editing process is interrupted, for example, by an error during validation. You can use the value of the **EditMode** property to determine whether you should use the **Update** or **CancelUpdate** method.

You can also check to see if the **LockEdits** property setting is **True** and the **EditMode** property setting is **dbEditInProgress** to determine whether the current page is locked.

Filter Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproFilterC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproFilterX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproFilterA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproFilterS"}

Sets or returns a value that determines the records included in a subsequently opened **Recordset** object (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **String** data type that contains the WHERE clause of an SQL statement without the reserved word WHERE.

Remarks

Use the **Filter** property to apply a filter to a dynaset-, snapshot-, or forward-only–type **Recordset** object.

You can use the **Filter** property to restrict the records returned from an existing object when a new **Recordset** object is opened based on an existing **Recordset** object.

In many cases, it's faster to open a new **Recordset** object by using an SQL statement that includes a WHERE clause.

Use the U.S. date format (month-day-year) when you filter fields containing dates, even if you're not using the U.S. version of the Microsoft Jet database engine (in which case you must assemble any dates by concatenating strings, for example, `strMonth & "-" & strDay & "-" & strYear`). Otherwise, the data may not be filtered as you expect.

If you set the property to a string concatenated with a non-integer value, and the system parameters specify a non-U.S. decimal character such as a comma (for example, `strFilter = "PRICE > " & lngPrice`, and `lngPrice = 125,50`), an error occurs when you try to open the next **Recordset**. This is because during concatenation, the number will be converted to a string using your system's default decimal character, and Microsoft Jet SQL only accepts U.S. decimal characters.

Foreign Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproForeignC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproForeignX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproForeignA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproForeignS"}

Returns a value that indicates whether an **Index** object represents a foreign key in a table (Microsoft Jet workspaces only).

Return Values

The return value is a **Boolean** data type that returns **True** if the **Index** object represents a foreign key.

Remarks

A foreign key consists of one or more fields in a foreign table that uniquely identify all rows in a primary table.

The Microsoft Jet database engine creates an **Index** object for the foreign table and sets the **Foreign** property when you create a relationship that enforces referential integrity.

ForeignName Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproForeignNameC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproForeignNameX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproForeignNameA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproForeignNameS"}

Sets or returns a value that specifies the name of the **Field** object in a foreign table that corresponds to a field in a primary table for a relationship (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **String** data type that evaluates to the name of a **Field** in the associated **TableDef** object's **Fields** collection.

If the **Relation** object isn't appended to the **Database**, but the **Field** is appended to the **Relation** object, the **ForeignName** property is read/write. Once the **Relation** object is appended to the database, the **ForeignName** property is read-only.

Remarks

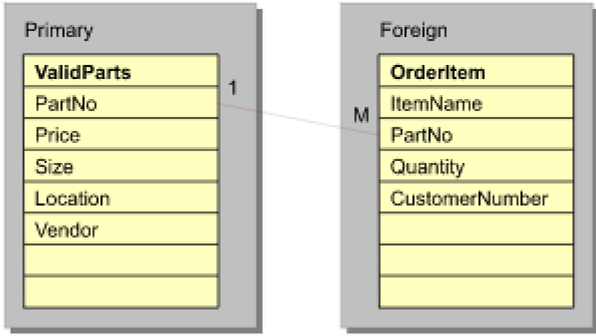
Only a **Field** object that belongs to the **Fields** collection of a **Relation** object can support the **ForeignName** property.

The **Name** and **ForeignName** property settings for a **Field** object specify the names of the corresponding fields in the primary and foreign tables of a relationship. The **Table** and **ForeignTable** property settings for a **Relation** object determine the primary and foreign tables of a relationship.

For example, if you had a list of valid part codes (in a field named PartNo) stored in a ValidParts table, you could establish a relationship with an OrderItem table such that if a part code were entered into the OrderItem table, it would have to already exist in the ValidParts table. If the part code didn't exist in the ValidParts table and you had not set the **Attributes** property of the **Relation** object to **dbRelationDontEnforce**, a trappable error would occur.

In this case, the ValidParts table is the foreign table, so the **ForeignTable** property of the **Relation** object would be set to ValidParts and the **Table** property of the **Relation** object would be set to OrderItem. The **Name** and **ForeignName** properties of the **Field** object in the **Relation** object's **Fields** collection would be set to PartNo.

The following illustration depicts the relation described above.



ForeignTable Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproForeignTableC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproForeignTableX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproForeignTableA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproForeignTableS"}

Sets or returns the name of the foreign table in a relationship (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **String** data type that evaluates to the name of a table in the **Database** object's **TableDefs** collection. This property is read/write for a new **Relation** object not yet appended to a collection and read-only for an existing **Relation** object in the **Relations** collection.

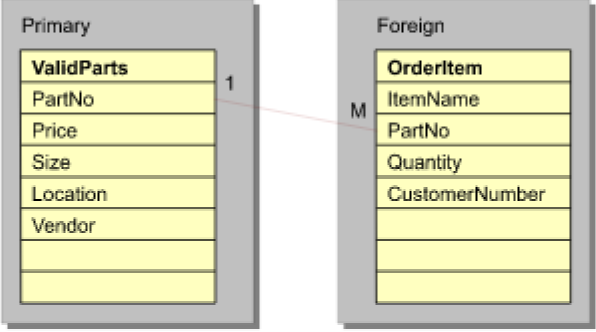
Remarks

The **ForeignTable** property setting of a **Relation** object is the **Name** property setting of the **TableDef** or **QueryDef** object that represents the foreign table or query; the **Table** property setting is the **Name** property setting of the **TableDef** or **QueryDef** object that represents the primary table or query.

For example, if you had a list of valid part codes (in a field named PartNo) stored in a ValidParts table, you could establish a relationship with an OrderItem table such that if a part code were entered into the OrderItem table, it would have to already be in the ValidParts table. If the part code didn't exist in the ValidParts table and you had not set the **Attributes** property of the **Relation** object to **dbRelationDontEnforce**, a trappable error would occur.

In this case, the ValidParts table is the primary table, so the **Table** property of the **Relation** object would be set to ValidParts and the **ForeignTable** property of the **Relation** object would be set to OrderItem. The **Name** and **ForeignName** properties of the **Field** object in the **Relation** object's **Fields** collection would be set to PartNo.

The following illustration depicts the relation described above.



HelpContext, HelpFile Properties

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproHelpContextC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproHelpContextX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproHelpContextA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproHelpContextS"}

- **HelpContext**—returns a context ID, as a **Long** variable, for a topic in a Microsoft Windows Help file.
- **HelpFile**—returns a **String** that is a fully qualified path to the Help file.

Remarks

If you specify a Microsoft Windows Help file in **HelpFile**, you can use the **HelpContext** property to automatically display the Help topic it identifies.

Note You should write procedures in your application to handle typical errors. When programming with an object, you can use the Help supplied by the object's Help file to improve the quality of your error handling, or to display a meaningful message to your user if the error is not recoverable.

IgnoreNulls Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproIgnoreNullsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproIgnoreNullsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproIgnoreNullsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproIgnoreNullsS"}

Sets or returns a value that indicates whether records that have **Null** values in their index fields have index entries (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **Boolean** that is **True** if the fields with **Null** values don't have an index entry. This property is read/write for a new **Index** object not yet appended to a collection and read-only for an existing **Index** object in an **Indexes** collection.

Remarks

To speed up the process of searching for records, you can define an index for a field. If you allow **Null** entries in an indexed field and expect many of the entries to be **Null**, you can set the **IgnoreNulls** property for the **Index** object to **True** to reduce the amount of storage space that the index uses.

The **IgnoreNulls** property setting and the **Required** property setting together determine whether a record with a **Null** index value has an index entry.

If IgnoreNulls is	And Required is	Then
True	**False**	A **Null** value is allowed in the index field; no index entry added.
False	**False**	A **Null** value is allowed in the index field; index entry added.
True or **False**	**True**	A **Null** value isn't allowed in the index field; no index entry added.

Index Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproIndexX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproIndexA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproIndexS"}

Sets or returns a value that indicates the name of the current **Index** object in a table-type **Recordset** object (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **String** data type that evaluates to the name of an **Index** object in the **Indexes** collection of the **Tabledef** or table-type **Recordset** object's **TableDef** object.

Remarks

Records in base tables aren't stored in any particular order. Setting the **Index** property changes the order of records returned from the database; it doesn't affect the order in which the records are stored.

The specified **Index** object must already be defined. If you set the **Index** property to an **Index** object that doesn't exist or if the **Index** property isn't set when you use the **Seek** method, a trappable error occurs.

Examine the **Indexes** collection of a **TableDef** object to determine what **Index** objects are available to table-type **Recordset** objects created from that **TableDef** object.

You can create a new index for the table by creating a new **Index** object, setting its properties, appending it to the **Indexes** collection of the underlying **TableDef** object, and then reopening the **Recordset** object.

Records returned from a table-type **Recordset** object can be ordered only by the indexes defined for the underlying **TableDef** object. To sort records in some other order, you can open a dynaset-, snapshot-, or forward-only–type **Recordset** object by using an SQL statement with an ORDER BY clause.

Notes

- You don't have to create indexes for tables. With large, unindexed tables, accessing a specific record or creating a **Recordset** object can take a long time. On the other hand, creating too many indexes slows down update, append, and delete operations because all indexes are automatically updated.
- Records read from tables without indexes are returned in no particular sequence.
- The **Attributes** property of each **Field** object in the **Index** object determines the order of records and consequently determines the access techniques to use for that index.
- A unique index helps optimize finding records.
- Indexes don't affect the physical order of a base table – indexes affect only how the records are accessed by the table-type **Recordset** object when a particular index is chosen or when **Recordset** is opened.

Inherit Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproInheritC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproInheritX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproInheritA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproInheritS"}

Sets or returns a value that indicates whether new **Document** objects will inherit a default **Permissions** property setting (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **Boolean** data type. If you set the property to **True**, **Document** objects inherit a default **Permissions** property setting.

Remarks

Use the **Inherit** property in conjunction with the **Permissions** property to define what permissions new documents will automatically have when they're created. If you set the **Inherit** property to **True**, and then set a permission on a container, then whenever a new document is created in that container, that permission will be set on the new document. This is a very convenient way of presetting permissions on an object.

Setting the **Inherit** property will not affect existing documents in the container − you can't modify all the permissions on all existing documents in a container by setting the **Inherit** property and a new permission. It will affect only new documents that are created after the **Inherit** property is set.

Inherited Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproInheritedC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproInheritedX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproInheritedA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproInheritedS"}

Returns a value that indicates whether a **Property** object is inherited from an underlying object.

Return Values

The return value is a **Boolean** data type that is **True** if the **Property** object is inherited. For built-in **Property** objects that represent predefined properties, the only possible return value is **False**. This property is always **False** in an ODBCDirect workspace.

Remarks

You can use the **Inherited** property to determine whether a user-defined **Property** was created for the object it applies to, or whether the **Property** was inherited from another object. For example, suppose you create a new **Property** for a **QueryDef** object and then open a **Recordset** object from the **QueryDef** object. This new **Property** will be part of the **Recordset** object's **Properties** collection, and its **Inherited** property will be set to **True** because the property was created for the **QueryDef** object, not the **Recordset** object.

IniPath Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproIniPathC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproIniPathX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproIniPathA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproIniPathS"}

Sets or returns information about the Windows Registry key that contains values for the Microsoft Jet database engine (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **String** data type that points to a user-supplied portion of the Windows Registry key containing Microsoft Jet database engine settings or parameters needed for installable ISAM databases.

Remarks

You can configure the Microsoft Jet engine with the Windows Registry. You can use the Registry to set options, such as installable ISAM DLLs.

For this option to have any effect, you must set the **IniPath** property before your application invokes any other DAO code. The scope of this setting is limited to your application and can't be changed without restarting your application.

You also use the Registry to provide initialization parameters for some installable ISAM database drivers. For example, to use Paradox version 4.0, set the **IniPath** property to a part of the Registry containing the appropriate parameters.

This property recognizes either HKEY_LOCAL_MACHINE or HKEY_LOCAL_USER. If no root key is supplied, the default is HKEY_LOCAL_MACHINE.

Microsoft Jet versions 2.5 or earlier kept initialization information in .ini files.

IsolateODBCTrans Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproIsolateODBCTransC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproIsolateODBCTransX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproIsolateODBCTransA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproIsolateODBCTransS"}

Sets or returns a value that indicates whether multiple transactions that involve the same Microsoft Jet-connected ODBC data source are isolated (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **Boolean** data type that is **True** if you want to isolate transactions involving the same ODBC (Open Database Connectivity) connection. **False** (the default) will allow multiple transactions involving the same ODBC connection.

Remarks

In some situations, you need to have multiple simultaneous transactions pending on the same ODBC connection. To do this, you need to open a separate **Workspace** for each transaction. Although each **Workspace** can have its own ODBC connection to the database, this slows system performance. Because transaction isolation isn't usually required, ODBC connections from multiple **Workspace** objects opened by the same user are shared by default.

Some ODBC servers, such as Microsoft SQL Server, don't allow simultaneous transactions on a single connection. If you need to have more than one transaction at a time pending against such a database, set the **IsolateODBCTrans** property to **True** on each **Workspace** as soon as you open it. This forces a separate ODBC connection for each **Workspace**.

LastModified Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproLastModifiedC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproLastModifiedX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproLastModifiedA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproLastModifiedS"}

Returns a bookmark indicating the most recently added or changed record.

Return Values

The return value is a **Variant** array of Byte data.

Remarks

You can use the **LastModified** property to move to the most recently added or updated record. Use the **LastModified** property with table- and dynaset-type **Recordset** objects. A record must be added or modified in the **Recordset** object itself in order for the **LastModified** property to have a value.

LockEdits Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproLockEditsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproLockEditsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproLockEditsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproLockEditsS"}

Sets or returns a value indicating the type of locking that is in effect while editing.

Settings and Return Values

The setting or return value is a **Boolean** that indicates the type of locking, as specified in the following table.

Value	Description
True	Default. Pessimistic locking is in effect. The 2K page containing the record you're editing is locked as soon as you call the **Edit** method.
False	Optimistic locking is in effect for editing. The 2K page containing the record is not locked until the **Update** method is executed.

Remarks

You can use the **LockEdits** property with updatable **Recordset** objects.

If a page is locked, no other user can edit records on the same page. If you set **LockEdits** to **True** and another user already has the page locked, an error occurs when you use the **Edit** method. Other users can read data from locked pages.

If you set the **LockEdits** property to **False** and later use the **Update** method while another user has the page locked, an error occurs. To see the changes made to your record by another user, use the **Move** method with 0 as the argument; however, if you do this, you will lose your changes.

When working with Microsoft Jet-connected ODBC data sources, the **LockEdits** property is always set to **False**, or optimistic locking. The Microsoft Jet database engine has no control over the locking mechanisms used in external database servers.

Note You can preset the value of **LockEdits** when you first open the **Recordset** by setting the *lockedits* argument of the **OpenRecordset** method. Setting the *lockedits* argument to **dbPessimistic** will set the **LockEdits** property to **True**, and setting *lockedits* to any other value will set the **LockEdits** property to **False.**

LoginTimeout Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproLoginTimeoutC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproLoginTimeoutX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproLoginTimeoutA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproLoginTimeoutS"}

Sets or returns the number of seconds before an error occurs when you attempt to log on to an ODBC database.

Settings and Return Values

The setting or return value is an **Integer** representing the number of seconds before a login timeout error occurs. The default **LoginTimeout** property setting is 20 seconds. When the **LoginTimeout** property is set to 0, no timeout occurs.

Remarks

When you're attempting to log on to an ODBC database, such as Microsoft SQL Server, the connection can fail as a result of network errors or because the server isn't running. Rather than waiting for the default 20 seconds to connect, you can specify how long to wait before raising an error. Logging on to the server happens implicitly as part of a number of different events, such as running a query on an external server database.

You can use **LoginTimeout** on the **DBEngine** object in both Microsoft Jet and ODBCDirect workspaces. You can use **LoginTimeout** on the **Workspace** object only in ODBCDirect workspaces. Setting the property to -1 on a **Workspace** will default to the current setting of **DBEngine.LoginTimeout**. You can change this property in a **Workspace** at any time, and the new setting will take effect with the next **Connection** or **Database** object opened.

The default value is determined by the ODBC driver. In a Microsoft Jet workspace, you can override the driver’s default value by creating a new “ODBC” key in the Registry path **\HKEY_LOCAL_MACHINE\SOFTWARE\Jet\3.5**\, creating a **LoginTimeout** parameter in this key, and setting the value as desired.

LogMessages Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproLogMessagesC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproLogMessagesX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproLogMessagesA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproLogMessagesS"}

Sets or returns a value that specifies if the messages returned from a Microsoft Jet-connected ODBC data source are recorded (Microsoft Jet workspaces only).

Note Before you can set or get the value of the **LogMessages** property, you must create the **LogMessages** property with the **CreateProperty** method, and append it to the **Properties** collection of a **QueryDef** object.

Settings and Return Values

The setting or return value is a **Boolean** that is **True** if ODBC-generated messages are recorded.

Remarks

Some pass-through queries can return messages in addition to data. If you set the **LogMessages** property to **True**, the Microsoft Jet database engine creates a table that contains returned messages. The table name is the user name concatenated with a hyphen (-) and a sequential number starting at 00. For example, because the default user name is Admin, the tables returned would be named Admin-00, Admin-01, and so on.

If you expect the query to return messages, create and append a user-defined **LogMessages** property for the **QueryDef** object, and set its type to **Boolean** and its value to **True**.

Once you've processed the results from these tables, you may want to delete them from the database along with the temporary query used to create them.

Name Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproNameC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproNameX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproNameA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproNameS"}

Sets or returns a user-defined name for a DAO object. For an object not appended to a collection, this property is read/write.

Settings and Return Values

The setting or return value is a **String** that specifies a name. The name must start with a letter. The maximum number of characters depends on the type of object **Name** applies to, as shown in Remarks. It can include numbers and underscore characters (_) but can't include punctuation or spaces.

Remarks

TableDef, **QueryDef**, **Field**, **Index**, **User**, and **Group** objects can't share the same name with any object in the same collection.

The **Name** property of a **Recordset** object opened by using an SQL statement is the first 256 characters of the SQL statement.

You can use an object's **Name** property with the Visual Basic for Applications **Dim** statement in code to create other instances of the object.

Note For many of the DAO objects, the **Name** property reflects the name as known to the **Database** object, as in the name of a **TableDef**, **Field**, or **QueryDef** object. There is no direct link between the name of the DAO object and the object variable used to reference it.

The read/write usage of the **Name** property depends on the type of object it applies to, and whether or not the object has been appended to a collection. In an ODBCDirect workspace, the **Name** property of an appended object is always read-only. The following table indicates whether the **Name** property in a Microsoft Jet workspace is read/write or read-only for an object that is appended to a collection (unless otherwise noted), and also indicates its maximum length in cases where it is read/write.

Object	Usage	Maximum length
Container	Read-only	
Connection	Read-only	
Database	Read-only	
Document	Read-only	
Field		
Unappended	Read/write	64
Appended to **Index**	Read-only	
Appended to **QueryDef**	Read-only	
Appended to **Recordset**	Read-only	
Appended to **TableDef** (native)	Read/write	64
Appended to **TableDef** (linked)	Read-only	
Appended to **Relation**	Read-only	
Group		
Unappended	Read/write	20
Appended	Read-only	

Index		
Unappended	Read/write	64
Appended	Read-only	
Parameter	Read-only	
Property		
Unappended	Read/write	64
Appended	Read-only	
Built-in	Read-only	
QueryDef		
Unappended	Read/write	64
Temporary	Read-only	
Appended	Read/write	64
Recordset	Read-only	
Relation		
Unappended	Read/write	64
Appended	Read-only	
TableDef	Read/write	64
User		
Unappended	Read/write	20
Appended	Read-only	
Workspace		
Unappended	Read/write	20
Appended	Read-only	

NoMatch Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproNoMatchC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproNoMatchX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproNoMatchA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproNoMatchS"}

Indicates whether a particular record was found by using the **Seek** method or one of the **Find** methods (Microsoft Jet workspaces only).

Return Values

The return value is a **Boolean** that is **True** if the desired record was not found. When you open or create a **Recordset** object, its **NoMatch** property is set to **False**.

Remarks

To locate a record, use the **Seek** method on a table-type **Recordset** object or one of the **Find** methods on a dynaset- or snapshot-type **Recordset** object. Check the **NoMatch** property setting to see whether the record was found.

If the **Seek** or **Find** method is unsuccessful and the **NoMatch** property is **True**, the current record will no longer be valid. Be sure to obtain the current record's bookmark before using the **Seek** method or a **Find** method if you'll need to return to that record.

Note Using any of the **Move** methods on a **Recordset** object won't affect its **NoMatch** property setting.

Number Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproNumberC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproNumberX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproNumberA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproNumberS"}

Returns a numeric value specifying an error. The **Number** property is the **Error** object's default property.

Return Values

The return value is a **Long** data type that represents an error number.

Remarks

Use the **Number** property to determine the error that occurred. The value of the property corresponds to a unique trap number that corresponds to an error condition. For a complete list of all trap numbers and error conditions, see Trappable Microsoft Jet and DAO Errors.

ODBCTimeout Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproODBCTimeoutC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproODBCTimeoutX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproODBCTimeoutA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproODBCTimeoutS"}

Indicates the number of seconds to wait before a timeout error occurs when a **QueryDef** is executed on an ODBC database.

Settings and Return Values

The setting or return value is an **Integer** representing the number of seconds to wait before a timeout error occurs.

When the **ODBCTimeout** property is set to -1, the timeout defaults to the current setting of the **QueryTimeout** property of the **Connection** or **Database** object that contains the **QueryDef**. When the **ODBCTimeout** property is set to 0, no timeout error occurs.

Remarks

When you're using an ODBC database, such as Microsoft SQL Server, delays can occur because of network traffic or heavy use of the ODBC server. Rather than waiting indefinitely, you can specify how long to wait before returning an error.

Setting the **ODBCTimeout** property of a **QueryDef** object overrides the value specified by the **QueryTimeout** property of the **Connection** or **Database** object containing the **QueryDef**, but only for that **QueryDef** object.

Note In an ODBCDirect workspace, after setting **ODBCTimeout** to an explicit value you can reset it back to the default (i.e., -1) only once during the life of the **QueryDef** object. Otherwise, an error will occur.

OrdinalPosition Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproOrdinalPositionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproOrdinalPositionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproOrdinalPositionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproOrdinalPositionS"}

Sets or returns the relative position of a **Field** object within a **Fields** collection. For an object not yet appended to the **Fields** collection, this property is read/write.

Settings and Return Values

The setting or return value is an **Integer** that specifies the numeric order of fields. The default is 0.

Remarks

The availability of the **OrdinalPosition** property depends on the object that contains the **Fields** collection, as shown in the following table.

If the Fields collection belongs to a	Then OrdinalPosition is
Index object	Not supported
QueryDef object	Read-only
Recordset object	Read-only
Relation object	Not supported
TableDef object	Read/write

Generally, the ordinal position of an object that you append to a collection depends on the order in which you append the object. The first appended object is in the first position (0), the second appended object is in the second position (1), and so on. The last appended object is in ordinal position *count* – 1, where *count* is the number of objects in the collection as specified by the **Count** property setting.

You can use the **OrdinalPosition** property to specify an ordinal position for new **Field** objects that differs from the order in which you append those objects to a collection. This enables you to specify a field order for your tables, queries, and recordsets when you use them in an application. For example, the order in which fields are returned in a `SELECT *` query is determined by the current **OrdinalPosition** property values.

You can permanently reset the order in which fields are returned in recordsets by setting the **OrdinalPosition** property to any positive integer.

Two or more **Field** objects in the same collection can have the same **OrdinalPosition** property value, in which case they will be ordered alphabetically. For example, if you have a field named Age set to 4 and you set a second field named Weight to 4, Weight is returned after Age.

You can specify a number that is greater than the number of fields minus 1. The field will be returned in an order relative to the largest number. For example, if you set a field's **OrdinalPosition** property to 20 (and there are only 5 fields) and you've set the **OrdinalPosition** property for two other fields to 10 and 30, respectively, the field set to 20 is returned between the fields set to 10 and 30.

Note Even if the **Fields** collection of a **TableDef** has not been refreshed, the field order in a **Recordset** opened from the **TableDef** will reflect the **OrdinalPosition** data of the **TableDef** object. A table-type **Recordset** will have the same **OrdinalPosition** data as the underlying table, but any other type of **Recordset** will have new **OrdinalPosition** data (starting with 0) that follow the order determined by the **OrdinalPosition** data of the **TableDef**.

Owner Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproOwnerC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproOwnerX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproOwnerA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproOwnerS"}

Sets or returns a value that specifies the owner of the object (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **String** that evaluates to either the name of a **User** object in the **Users** collection or the name of a **Group** object in the **Groups** collection.

Remarks

The owner of an object has certain access privileges denied to other users. Any individual user account (represented by a **User** object) or group of user accounts (represented by a **Group** object) can change the **Owner** property setting at any time if it has the appropriate permissions.

Password Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproPasswordC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproPasswordX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproPasswordA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproPasswordS"}

Sets the password for a user account (Microsoft Jet workspaces only).

Settings

The setting is a **String** that can be up to 14 characters long and can include any characters except the ASCII character 0 (null). This property setting is write-only for new objects not yet appended to a collection, and is not available for existing objects.

Remarks

Set the **Password** property along with the **PID** property when you create a new **User** object.

Use the **NewPassword** method to change the **Password** property setting for an existing **User** object. To clear a password, set the *newpassword* argument of the **NewPassword** method to a zero-length string ("").

Passwords are case-sensitive.

Note If you don't have access permission, you can't change the password of any other user.

PercentPosition Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproPercentPositionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproPercentPositionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproPercentPositionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproPercentPositionS"}

Sets or returns a value indicating the approximate location of the current record in the **Recordset** object based on a percentage of the records in the **Recordset**.

Settings and Return Values

The setting or return value is a **Single** that is a number between 0.0 and 100.00.

Remarks

To indicate or change the approximate position of the current record in a **Recordset** object, you can check or set the **PercentPosition** property. When working with a dynaset- or snapshot-type **Recordset** object opened directly from a base table, first populate the **Recordset** object by moving to the last record before you set or check the **PercentPosition** property. If you use the **PercentPosition** property before fully populating the **Recordset** object, the amount of movement is relative to the number of records accessed as indicated by the **RecordCount** property setting. You can move to the last record by using the **MoveLast** method.

Note Using the **PercentPosition** property to move the current record to a specific record in a **Recordset** object isn't recommended—the **Bookmark** property is better suited for this task.

Once you set the **PercentPosition** property to a value, the record at the approximate position corresponding to that value becomes current, and the **PercentPosition** property is reset to a value that reflects the approximate position of the current record. For example, if your **Recordset** object contains only five records, and you set its **PercentPosition** property value to 77, the value returned from the **PercentPosition** property may be 80, not 77.

The **PercentPosition** property applies to all types of **Recordset** objects except for forward-only–type **Recordset** objects or **Recordset** objects opened from pass-through queries against remote databases.

You can use the **PercentPosition** property with a scroll bar on a form or text box to indicate the location of the current record in a **Recordset** object.

Permissions Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproPermissionsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproPermissionsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproPermissionsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproPermissionsS"}

Sets or returns a value that establishes the permissions for the user or group identified by the **UserName** property of a **Container** or **Document** object (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **Long** constant that establishes permissions. The following tables list the valid constants for the **Permissions** property of various DAO objects. Unless otherwise noted, all constants shown in all tables are valid for **Document** objects.

The following table lists possible values for **Container** objects other than Tables and Databases containers.

Constant	Description
dbSecNoAccess	The user doesn't have access to the object (not valid for **Document** objects).
dbSecFullAccess	The user has full access to the object.
dbSecDelete	The user can delete the object.
dbSecReadSec	The user can read the object's security-related information.
dbSecWriteSec	The user can alter access permissions.
dbSecWriteOwner	The user can change the **Owner** property setting.

The following tables lists the possible settings and return values for the Tables container.

Constant	Description
dbSecCreate	The user can create new documents (not valid for **Document** objects).
dbSecReadDef	The user can read the table definition, including column and index information.
dbSecWriteDef	The user can modify or delete the table definition, including column and index information.
dbSecRetrieveData	The user can retrieve data from the **Document** object.
dbSecInsertData	The user can add records.
dbSecReplaceData	The user can modify records.
dbSecDeleteData	The user can delete records.

The following tables lists the possible settings and return values for the Databases container.

Constant	Description
dbSecDBAdmin	The user can replicate a database and change the database password (not valid for **Document** objects).
dbSecDBCreate	The user can create new databases. This option is valid only on the

	Databases container in the workgroup information file (Systen.mdw). This constant isn't valid for **Document** objects.
dbSecDBExclusive	The user has exclusive access to the database.
dbSecDBOpen	The user can open the database.

Remarks

Use this property to establish or determine the type of read/write permissions the user has for a **Container** or **Document** object.

A **Document** object inherits the permissions for users from its **Container** object, provided the **Inherit** property of the **Container** object is set for those users or for a group to which the users belong. By setting a **Document** object's **Permissions** and **UserName** properties later, you can further refine the access control behavior of your object.

If you want to set or return permissions for a user that includes permissions inherited from any groups to which the user belongs, use the **AllPermissions** property.

PID Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproPIDC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproPIDX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproPIDA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproPIDS"}

Sets the personal identifier (PID) for either a group or a user account (Microsoft Jet workspaces only).

Settings

The setting is a **String** containing 4-20 alphanumeric characters. This property setting is write-only for new objects not yet appended to a collection, and is not available for existing objects.

Remarks

Set the **PID** property along with the **Name** property when you create a new **Group** object. Set the **PID** property along with the **Name** and **Password** properties when you create a new **User** object.

Primary Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproPrimaryC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproPrimaryX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproPrimaryA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproPrimaryS"}

Sets or returns a value that indicates whether an **Index** object represents a primary index for a table (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **Boolean** that is **True** if the **Index** object represents a primary index.

The **Primary** property setting is read/write for a new **Index** object not yet appended to a collection and read-only for an existing **Index** object in an **Indexes** collection. If the **Index** object is appended to the **TableDef** object but the **TableDef** object isn't appended to the **TableDefs** collection, the **Index** property is read/write.

Remarks

A primary index consists of one or more fields that uniquely identify all records in a table in a predefined order. Because the index field must be unique, the **Unique** property of the **Index** object is set to **True**. If the primary index consists of more than one field, each field can contain duplicate values, but each combination of values from all the indexed fields must be unique. A primary index consists of a key for the table and usually contains the same fields as the primary key.

Note You don't have to create indexes for tables, but in large, unindexed tables, accessing a specific record can take a long time. The **Attributes** property of each **Field** object in the **Index** object determines the order of records and consequently determines the access techniques to use for that index. When you create a new table in your database, it's a good idea to create an index on one or more fields that uniquely identify each record, and then set the **Primary** property of the **Index** object to **True**.

When you set a primary key for a table, the primary key is automatically defined as the primary index for the table.

QueryTimeout Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproQueryTimeoutC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproQueryTimeoutX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproQueryTimeoutA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproQueryTimeoutS"}

Sets or returns a value that specifies the number of seconds to wait before a timeout error occurs when a query is executed on an ODBC data source.

Settings and Return Values

The setting or return value is an **Integer** representing the number of seconds to wait. The default value is 60.

Remarks

When you're using an ODBC database, such as Microsoft SQL Server, there may be delays due to network traffic or heavy use of the ODBC server. Rather than waiting indefinitely, you can specify how long to wait.

When you use **QueryTimeout** with a **Connection** or **Database** object, it specifies a global value for all queries associated with the database. You can override this value for a specific query by setting the **ODBCTimeout** property of the particular **QueryDef** object.

In a Microsoft Jet workspace, you can override the default value by creating a new "ODBC" key in the Registry path **\HKEY_LOCAL_MACHINE\SOFTWARE\Jet\3.5**\, creating a **QueryTimeout** parameter in this key, and setting the value as desired.

RecordCount Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproRecordCountC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproRecordCountX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproRecordCountA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproRecordCountS"}

Returns the number of records accessed in a **Recordset** object, or the total number of records in a table-type **Recordset** or **TableDef** object.

Return Values

The return value is a **Long** data type.

Remarks

Use the **RecordCount** property to find out how many records in a **Recordset** or **TableDef** object have been accessed. The **RecordCount** property doesn't indicate how many records are contained in a dynaset-, snapshot-, or forward-only–type **Recordset** object until all records have been accessed. Once the last record has been accessed, the **RecordCount** property indicates the total number of undeleted records in the **Recordset** or **TableDef** object. To force the last record to be accessed, use the **MoveLast** method on the **Recordset** object. You can also use an SQL **Count** function to determine the approximate number of records your query will return.

Note Using the **MoveLast** method to populate a newly opened **Recordset** negatively impacts performance. Unless it is necessary to have an accurate **RecordCount** as soon as you open a **Recordset**, it's better to wait until you populate the **Recordset** with other portions of code before checking the **RecordCount** property.

As your application deletes records in a dynaset-type **Recordset** object, the value of the **RecordCount** property decreases. However, records deleted by other users aren't reflected by the **RecordCount** property until the current record is positioned to a deleted record. If you execute a transaction that affects the **RecordCount** property setting and you subsequently roll back the transaction, the **RecordCount** property won't reflect the actual number of remaining records.

The **RecordCount** property of a snapshot- or forward-only–type **Recordset** object isn't affected by changes in the underlying tables.

A **Recordset** or **TableDef** object with no records has a **RecordCount** property setting of 0.

When you work with linked **TableDef** objects, the **RecordCount** property setting is always –1.

Using the **Requery** method on a **Recordset** object resets the **RecordCount** property just as if the query were re-executed.

RecordsAffected Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproRecordsAffectedC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproRecordsAffectedX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproRecordsAffectedA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproRecordsAffectedS"}

Returns the number of records affected by the most recently invoked **Execute** method.

Return Values

The return value is a **Long** from 0 to the number of records affected by the most recently invoked **Execute** method on either a **Database** or **QueryDef** object.

Remarks

When you use the **Execute** method to run an action query from a **QueryDef** object, the **RecordsAffected** property will contain the number of records deleted, updated, or inserted.

When you use **RecordsAffected** in an ODBCDirect workspace, it will not return a useful value from an SQL DROP TABLE action query.

Required Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproRequiredC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproRequiredX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproRequiredA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproRequiredS"}

Sets or returns a value that indicates whether a **Field** object requires a non-**Null** value or whether all the fields in an **Index** object must have a value.

Settings and Return Values

The setting or return value is a **Boolean** that is **True** if a field can't contain a **Null** value.

For an object not yet appended to a collection, this property is read/write. For an **Index** object, this property setting is read-only for objects appended to **Indexes** collections in **Recordset** and **TableDef** objects.

Remarks

The availability of the **Required** property depends on the object that contains the **Fields** collection, as shown in the following table.

If the Fields collection belongs to a	Then Required is
Index object	Not supported
QueryDef object	Read-only
Recordset object	Read-only
Relation object	Not supported
TableDef object	Read/write

For a **Field** object, you can use the **Required** property along with the **AllowZeroLength**, **ValidateOnSet**, or **ValidationRule** property to determine the validity of the **Value** property setting for that **Field** object. If the **Required** property is set to **False**, the field can contain **Null** values as well as values that meet the conditions specified by the **AllowZeroLength** and **ValidationRule** property settings.

Note When you can set this property for either an **Index** object or a **Field** object, set it for the **Field** object. The validity of the property setting for a **Field** object is checked before that of an **Index** object.

Restartable Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproRestartableC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproRestartableX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproRestartableA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproRestartableS"}

Returns a value that indicates whether a **Recordset** object supports the **Requery** method, which re-executes the query on which the **Recordset** object is based.

Return Values

The return value is a **Boolean** data type that is **True** if the **Recordset** object supports the **Requery** method. Table-type **Recordset** objects always return **False**.

Remarks

Check the **Restartable** property before using the **Requery** method on a **Recordset** object. If the object's **Restartable** property is set to **False**, use the **OpenRecordset** method on the underlying **QueryDef** object to re-execute the query.

ReturnsRecords Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproReturnsRecordsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproReturnsRecordsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproReturnsRecordsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproReturnsRecordsS"}

Sets or returns a value that indicates whether an SQL pass-through query to an external database returns records.

Settings and Return Values

The setting or return value is a **Boolean** that is **True** (default) if a pass-through query returns records.

Remarks

Not all SQL pass-through queries to external databases return records. For example, an SQL UPDATE statement updates records without returning records, while an SQL SELECT statement does return records. If the query returns records, set the **ReturnsRecords** property to **True**; if the query doesn't return records, set the **ReturnsRecords** property to **False**.

Note You must set the **Connect** property before you set the **ReturnsRecords** property.

Size Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproSizeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproSizeX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproSizeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproSizeS"}

Sets or returns a value that indicates the maximum size, in bytes, of a **Field** object.

Settings and Return Values

The setting or return value is a constant that indicates the maximum size of a **Field** object. For an object not yet appended to the **Fields** collection, this property is read/write. The setting depends on the **Type** property setting of the **Field** object, as discussed under Remarks.

Remarks

For fields (other than Memo type fields) that contain character data, the **Size** property indicates the maximum number of characters that the field can hold. For numeric fields, the **Size** property indicates how many bytes of storage are required.

Use of the **Size** property depends on the object that contains the **Fields** collection to which the **Field** object is appended, as shown in the following table.

Object appended to	Usage
Index	Not supported
QueryDef	Read-only
Recordset	Read-only
Relation	Not supported
TableDef	Read-only

When you create a **Field** object with a data type other than Text, the **Type** property setting automatically determines the **Size** property setting; you don't need to set it. For a **Field** object with the Text data type, however, you can set **Size** to any integer up to the maximum text size (255 for Microsoft Jet databases). If you do not set the size, the field will be as large as the database allows.

For Long Binary and Memo **Field** objects, **Size** is always set to 0. Use the **FieldSize** property of the **Field** object to determine the size of the data in a specific record. The maximum size of a Long Binary or Memo field is limited only by your system resources or the maximum size that the database allows.

Sort Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproSortC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproSortX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproSortA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproSortS"}

Sets or returns the sort order for records in a **Recordset** object (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **String** that contains the ORDER BY clause of an SQL statement without the reserved words ORDER BY.

Remarks

You can use the **Sort** property with dynaset- and snapshot-type **Recordset** objects.

When you set this property for an object, sorting occurs when a subsequent **Recordset** object is created from that object. The **Sort** property setting overrides any sort order specified for a **QueryDef** object.

The default sort order is ascending (A to Z or 0 to 100).

The **Sort** property doesn't apply to table- or forward-only–type **Recordset** objects. To sort a table-type **Recordset** object, use the **Index** property.

Note In many cases, it's faster to open a new **Recordset** object by using an SQL statement that includes the sorting criteria.

Source Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproSourceC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproSourceX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproSourceA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproSourceS"}

Returns the name of the object or application that originally generated the error.

Return Values

The return value is a **String** representing the object or application that generated the error.

Remarks

The **Source** property value is usually the object's class name or programmatic ID. Use the **Source** property to provide your users with information when your code is unable to handle an error generated in an object in another application.

For example, if you access Microsoft Excel and it generates a "Division by zero" error, Microsoft Excel sets **Error.Number** to the Microsoft Excel code for that error and sets the **Source** property to `Excel.Application`. Note that if the error is generated in another object called by Microsoft Excel, Microsoft Excel intercepts the error and still sets **Error.Number** to the Microsoft Excel code. However, the other **Error** object properties (including **Source**) will retain the values as set by the object that generated the error. The **Source** property always contains the name of the object that originally generated the error.

Based on all of the error documentation, you can write code that will handle the error appropriately. If your error handler fails, you can use the **Error** object information to describe the error to your user, using the **Source** property and the other **Error** properties to give the user information about which object originally caused the error, the description of the error, and so forth.

Note The **On Error Resume Next** construct may be preferable to **On Error GoTo** when dealing with errors generated during access to other objects. Checking the **Error** object property after each interaction with an object removes ambiguity about which object your code was accessing when the error occurred. Thus, you can be sure which object placed the error code in **Error.Number**, as well as which object originally generated the error (**Error.Source**).

SourceField, SourceTable Properties

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproSourceFieldC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproSourceFieldX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproSourceFieldA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproSourceFieldS"}

- **SourceField** — returns a value that indicates the name of the field that is the original source of the data for a **Field** object.
- **SourceTable** — returns a value that indicates the name of the table that is the original source of the data for a **Field** object.

Return Values

The return value is a **String** specifying the name of the field or table that is the source of data.

Remarks

For a **Field** object, use of the **SourceField** and **SourceTable** properties depends on the object that contains the **Fields** collection that the **Field** object is appended to, as shown in the following table.

Object appended to	Usage
Index	Not supported
QueryDef	Read-only
Recordset	Read-only
Relation	Not supported
TableDef	Read-only

These properties indicate the original field and table names associated with a **Field** object. For example, you could use these properties to determine the original source of the data in a query field whose name is unrelated to the name of the field in the underlying table.

Note The **SourceTable** property will not return a meaningful table name if used on a **Field** object in the **Fields** collection of a table-type **Recordset** object.

SourceTableName Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproSourceTableNameC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproSourceTableNameX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproSourceTableNameA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproSourceTableNameS"}

Sets or returns a value that specifies the name of a linked table or the name of a base table (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **String** that specifies a table name. For a base table, the setting is a zero-length string (""). This property setting is read-only for a base table and read/write for a linked table or an object not appended to a collection.

SQL Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproSQLC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproSQLX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproSQLA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproSQLS"}

Sets or returns the SQL statement that defines the query executed by a **QueryDef** object.

Settings and Return Values

The setting or return value is a **String** that contains an SQL statement.

Remarks

The **SQL** property contains the SQL statement that determines how records are selected, grouped, and ordered when you execute the query. You can use the query to select records to include in a **Recordset** object. You can also define action queries to modify data without returning records.

The SQL syntax used in a query must conform to the SQL dialect of the query engine, which is determined by the type of workspace. In a Microsoft Jet workspace, use the Microsoft Jet SQL dialect, unless you create an SQL pass-through query, in which case you should use the dialect of the server. In an ODBCDirect workspace, use the SQL dialect of the server.

Note You can send DAO queries to a variety of different database servers with ODBCDirect, and different servers will recognize slightly different dialects of SQL. Therefore, context-sensitive Help is no longer provided for Microsoft Jet SQL, although online Help for Microsoft Jet SQL is still included through the Help menu. Be sure to check the appropriate reference documentation for the SQL dialect of your database server when using either ODBCDirect connections or pass-through queries in Microsoft Jet-connected client/server applications.

If the SQL statement includes parameters for the query, you must set these before execution. Until you reset the parameters, the same parameter values are applied each time you execute the query.

In an ODBCDirect workspace, you can also use the **SQL** property to execute a prepared statement on the server. For example, setting the **SQL** property to the following string will execute a prepared statement named “GetData” with one parameter on a Microsoft SQL Server back-end.

```
"{call GetData (?)}"
```

In a Microsoft Jet workspace, using a **QueryDef** object is the preferred way to perform SQL pass-through operations on Microsoft Jet-connected ODBC data sources. By setting the **QueryDef** object's **Connect** property to an ODBC data source, you can use non–Microsoft-Jet-database SQL in the query to be passed to the external server. For example, you can use TRANSACT SQL statements (with Microsoft SQL Server or Sybase SQL Server databases), which the Microsoft Jet database engine would otherwise not process.

Note If you set the property to a string concatenated with a non-integer value, and the system parameters specify a non-U.S. decimal character such as a comma (for example, `strSQL = "PRICE > " & lngPrice`, and `lngPrice = 125,50`), an error will result when you try to execute the **QueryDef** object in a Microsoft Jet database. This is because during concatenation, the number will be converted to a string using your system's default decimal character, and Microsoft Jet SQL only accepts U.S. decimal characters.

Table Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproTableC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproTableX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproTableA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproTableS"}

Indicates the name of a **Relation** object's primary table. This should be equal to the **Name** property setting of a **TableDef** or **QueryDef** object (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **String** that evaluates to the name of a table in the **TableDefs** collection or query in the **QueryDefs** collection. The **Table** property setting is read/write for a new **Relation** object not yet appended to a collection and read-only for an existing **Relation** object in a **Relations** collection.

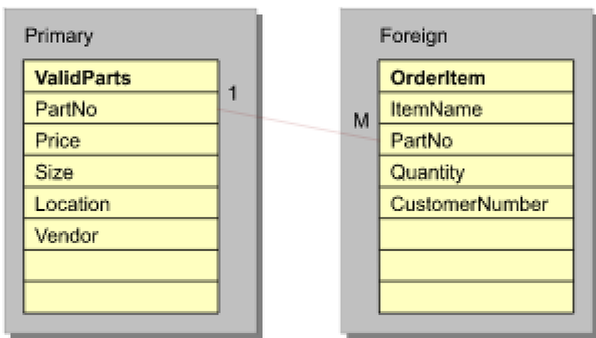
Remarks

Use the **Table** property with the **ForeignTable** property to define a **Relation** object, which represents the relationship between fields in two tables or queries. Set the **Table** property to the **Name** property setting of the primary **TableDef** or **QueryDef** object, and set the **ForeignTable** property to the **Name** property setting of the foreign (referencing) **TableDef** or **QueryDef** object. The **Attributes** property determines the type of relationship between the two objects.

For example, if you had a list of valid part codes (in a field named PartNo) stored in a ValidParts table, you could establish a one-to-many relationship with an OrderItem table such that if a part code were entered into the OrderItem table, it would have to already be in the ValidParts table. If the part code didn't exist in the ValidParts table and you had not set the **Attributes** property of the **Relation** object to **dbRelationDontEnforce**, a trappable error would occur.

In this case, the ValidParts table is the primary table, so the **Table** property of the **Relation** object would be set to ValidParts and the **ForeignTable** property of the **Relation** object would be set to OrderItem. The **Name** and **ForeignName** properties of the **Field** object in the **Relation** object's **Fields** collection would be set to PartNo.

The following illustration depicts this relation.



Transactions Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproTransactionsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproTransactionsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproTransactionsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproTransactionsS"}

Returns a value that indicates whether an object supports transactions.

Return Values

The return value is a **Boolean** data type that is **True** if the object supports transactions.

Remarks

In an ODBCDirect workspace, the **Transactions** property is available on Connection and Database objects, and indicates whether or not the ODBC driver you are using supports transactions.

In a Microsoft Jet workspace, you can also use the **Transactions** property with dynaset- or table-type **Recordset** objects. Snapshot- and forward-only–type **Recordset** objects always return **False**.

If a dynaset- or table-type **Recordset** is based on a Microsoft Jet database engine table, the **Transactions** property is **True** and you can use transactions. Other database engines may not support transactions. For example, you can't use transactions in a dynaset-type **Recordset** object based on a Paradox table.

Check the **Transactions** property before using the **BeginTrans** method on the **Recordset** object's **Workspace** object to make sure that transactions are supported. Using the **BeginTrans**, **CommitTrans**, or **Rollback** methods on an unsupported object has no effect.

Type Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproTypeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproTypeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproTypeS"}

Sets or returns a value that indicates the operational type or data type of an object.

Settings and Return Values

The setting or return value is a constant that indicates an operational or data type. For a **Field** or **Property** object, this property is read/write until the object is appended to a collection or to another object, after which it's read-only. For a **QueryDef**, **Recordset**, or **Workspace** object, the property setting is read-only. For a **Parameter** object in a Microsoft Jet workspace the property is read-only, while in an ODBCDirect workspace the property is always read-write.

For a **Field**, **Parameter**, or **Property** object, the possible settings and return values are described in the following table.

Constant	Description
dbBigInt	Big Integer
dbBinary	Binary
dbBoolean	Boolean
dbByte	Byte
dbChar	Char
dbCurrency	Currency
dbDate	Date/Time
dbDecimal	Decimal
dbDouble	Double
dbFloat	Float
dbGUID	GUID
dbInteger	Integer
dbLong	Long
dbLongBinary	Long Binary (OLE Object)
dbMemo	Memo
dbNumeric	Numeric
dbSingle	Single
dbText	Text
dbTime	Time
dbTimeStamp	Time Stamp
dbVarBinary	VarBinary

For a **QueryDef** object, the possible settings and return values are shown in the following table.

Constant	Query type
dbQAction	Action
dbQAppend	Append
dbQCompound	Compound
dbQCrosstab	Crosstab
dbQDDL	Data-definition
dbQDelete	Delete
dbQMakeTable	Make-table

dbQProcedure	Procedure (ODBCDirect workspaces only)
dbQSelect	Select
dbQSetOperation	Union
dbQSPTBulk	Used with **dbQSQLPassThrough** to specify a query that doesn't return records (Microsoft Jet workspaces only).
dbQSQLPassThrough	Pass-through (Microsoft Jet workspaces only)
dbQUpdate	Update

Note To create an SQL pass-through query in a Microsoft Jet workspace, you don't need to explicitly set the **Type** property to **dbQSQLPassThrough**. The Microsoft Jet database engine automatically sets this when you create a **QueryDef** object and set the **Connect** property.

For a **Recordset** object, the possible settings and return values are as follows.

Constant	Recordset type
dbOpenTable	Table (Microsoft Jet workspaces only)
dbOpenDynamic	Dynamic (ODBCDirect workspaces only)
dbOpenDynaset	Dynaset
dbOpenSnapshot	Snapshot
dbOpenForwardOnly	Forward-only

For a **Workspace** object, the possible settings and return values are as follows.

Constant	Workspace type
dbUseJet	The **Workspace** is connected to the Microsoft Jet database engine.
dbUseODBC	The **Workspace** is connected to an ODBC data source.

Remarks

When you append a new **Field**, **Parameter**, or **Property** object to the collection of an **Index**, **QueryDef**, **Recordset**, or **TableDef** object, an error occurs if the underlying database doesn't support the data type specified for the new object.

Unique Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproUniqueC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproUniqueX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproUniqueA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproUniqueS"}

Sets or returns a value that indicates whether an **Index** object represents a unique (key) index for a table (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **Boolean** that is **True** if the **Index** object represents a unique index. For an **Index** object, this property setting is read/write until the object is appended to a collection, after which it's read-only.

Remarks

A unique index consists of one or more fields that logically arrange all records in a table in a unique, predefined order. If the index consists of one field, values in that field must be unique for the entire table. If the index consists of more than one field, each field can contain duplicate values, but each combination of values from all the indexed fields must be unique.

If both the **Unique** and **Primary** properties of an **Index** object are set to **True**, the index is unique and primary: It uniquely identifies all records in the table in a predefined, logical order. If the **Primary** property is set to **False**, the index is a secondary index. Secondary indexes (both key and nonkey) logically arrange records in a predefined order without serving as an identifier for records in the table.

Notes

- You don't have to create indexes for tables, but in large, unindexed tables, accessing a specific record can take a long time.
- Records retrieved from tables without indexes are returned in no particular sequence.
- The **Attributes** property of each **Field** object in the **Index** object determines the order of records and consequently determines the access techniques to use for that **Index** object.
- A unique index helps optimize finding records.
- Indexes don't affect the physical order of a base table – indexes affect only how the records are accessed by the table-type **Recordset** object when a particular index is chosen or when the Microsoft Jet database engine creates **Recordset** objects.

Updatable Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproUpdatableC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproUpdatableX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproUpdatableA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproUpdatableS"}

Returns a value that indicates whether you can change a DAO object.

Return Values

The return value is a **Boolean** data type that is **True** if the object can be changed or updated. (Snapshot- and forward-only–type **Recordset** objects always return **False**.)

Remarks

Depending on the object, if the **Updatable** property setting is **True**, the associated statement in the following table is true.

Object	Type indicates
Database	The object can be changed
QueryDef	The query definition can be changed
Recordset	The records can be updated
TableDef	The table definition can be changed

The **Updatable** property setting is always **True** for a newly created **TableDef** object and **False** for a linked **TableDef** object. A new **TableDef** object can be appended only to a database for which the current user has write permission.

Many types of objects can contain fields that can't be updated. For example, you can create a dynaset-type **Recordset** object in which only some fields can be changed. These fields can be fixed or contain data that increments automatically, or the dynaset can result from a query that combines updatable and nonupdatable tables.

If the object contains only read-only fields, the value of the **Updatable** property is **False**. When one or more fields are updatable, the property's value is **True**. You can edit only the updatable fields. A trappable error occurs if you try to assign a new value to a read-only field.

The **Updatable** property of a **QueryDef** object is set to **True** if the query definition can be updated, even if the resulting **Recordset** object isn't updatable.

Because an updatable object can contain read-only fields, check the **DataUpdatable** property of each field in the **Fields** collection of a **Recordset** object before you edit a record.

UserName Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproUserNameC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproUserNameX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproUserNameA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproUserNameS"}

Sets or returns a value that represents a user, a group of users, or the owner of a **Workspace** object.

Settings and Return Values

The setting or return value is a **String** that evaluates to the name of a user. In a Microsoft Jet workspace, this represents a **User** object in the **Users** collection or a **Group** object in the **Groups** collection. For Microsoft Jet **Container** and **Document** objects, this property setting is read/write. For all **Workspace** objects, this property setting is read-only.

Remarks

Depending on the type of object, the **UserName** property represents the following.

- The owner of a **Workspace** object.
- A user or group of users when you manipulate the access permissions of a **Container** object or a **Document** object (Microsoft Jet workspaces only).

To find or set the permissions for a particular user or group of users, first set the **UserName** property to the user or group name that you want to examine. Then check the **Permissions** property setting to determine what permissions that user or group of users has, or set the **Permissions** property to change the permissions.

For a **Workspace** object, check the **UserName** property setting to determine the owner of the **Workspace** object. Set the **UserName** property to establish the owner of the **Workspace** object before you append the object to the **Workspaces** collection.

V1xNullBehavior Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproV1xNullBehaviorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproV1xNullBehaviorX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproV1xNullBehaviorA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproV1xNullBehaviorS"}

Indicates whether zero-length strings ("") used in code to fill Text or Memo fields are converted to **Null**.

Settings and Return Values

The setting or return value is a **Boolean** that is **True** if zero-length strings are converted to **Null**.

Remarks

This property applies to Microsoft Jet database engine version 1.x databases that have been converted to Microsoft Jet database engine version 2.0 or 3.0 databases.

Note The Microsoft Jet database engine automatically creates this property when it converts a version 1.x database to a version 2.0 or 3.x database. A 2.0 database will retain this property when it is converted to a 3.x database.

If you change this property setting, you must close and then reopen the database for your change to take effect.

For fastest performance, modify code that sets any Text or Memo fields to zero-length strings so that the fields are set to **Null** instead, and remove the **V1xNullBehavior** property from the **Properties** collection.

ValidateOnSet Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproValidateOnSetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproValidateOnSetX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproValidateOnSetA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproValidateOnSetS"}

Sets or returns a value that specifies whether or not the value of a **Field** object is immediately validated when the object's **Value** property is set (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **Boolean** that can be one of the following values.

Value	Description
True	The validation rule specified by the **ValidationRule** property setting of the **Field** object is checked when you set the object's **Value** property.
False	(Default) Validate when the record is updated.

Only **Field** objects in **Recordset** objects support the **ValidateOnSet** property as read/write.

Remarks

Setting the **ValidateOnSet** property to **True** can be useful in a situation when a user is entering records that include substantial Memo data. Waiting until the **Update** call to validate the data can result in unnecessary time spent writing the lengthy Memo data to the database if it turns out that the data was invalid anyway because a validation rule was broken in another field.

ValidationRule Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproValidationRuleC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproValidationRuleX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproValidationRuleA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproValidationRuleS"}

Sets or returns a value that validates the data in a field as it's changed or added to a table (Microsoft Jet workspaces only).

Settings and Return Values

The settings or return values is a **String** that describes a comparison in the form of an SQL WHERE clause without the WHERE reserved word. For an object not yet appended to the **Fields** collection, this property is read/write. See Remarks for the more specific read/write characteristics of this property.

Remarks

The **ValidationRule** property determines whether or not a field contains valid data. If the data is not valid, a trappable run-time error occurs. The returned error message is the text of the **ValidationText** property, if specified, or the text of the expression specified by **ValidationRule**.

For a **Field** object, use of the **ValidationRule** property depends on the object that contains the **Fields** collection to which the **Field** object is appended.

Object appended to	Usage
Index	Not supported
QueryDef	Read-only
Recordset	Read-only
Relation	Not supported
TableDef	Read/write

For a **Recordset** object, use of the **ValidationRule** property is read-only. For a **TableDef** object, use of the **ValidationRule** property depends on the status of the **TableDef** object, as the following table shows.

TableDef	Usage
Base table	Read/write
Linked table	Read-only

Validation is supported only for databases that use the Microsoft Jet database engine.

The string expression specified by the **ValidationRule** property of a **Field** object can refer only to that **Field**. The expression can't refer to user-defined functions, SQL aggregate functions, or queries. To set a **Field** object's **ValidationRule** property when its **ValidateOnSet** property setting is **True**, the expression must successfully parse (with the field name as an implied operand) and evaluate to **True**. If its **ValidateOnSet** property setting is **False**, the **ValidationRule** property setting is ignored.

The **ValidationRule** property of a **Recordset** or **TableDef** object can refer to multiple fields in that object. The restrictions noted earlier in this topic for the **Field** object apply.

For a table-type **Recordset** object, the **ValidationRule** property inherits the **ValidationRule** property setting of the **TableDef** object that you use to create the table-type **Recordset** object.

For a **TableDef** object based on an linked table, the **ValidationRule** property inherits the **ValidationRule** property setting of the underlying base table. If the underlying base table doesn't support validation, the value of this property is a zero-length string ("").

Note If you set the property to a string concatenated with a non-integer value, and the system parameters specify a non-U.S. decimal character such as a comma (for example, `strRule =`

`"PRICE > " & lngPrice`, and `lngPrice = 125,50`), an error will result when your code attempts to validate any data. This is because during concatenation, the number will be converted to a string using your system's default decimal character, and Microsoft Jet SQL only accepts U.S. decimal characters.

ValidationText Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproValidationTextC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproValidationTextX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproValidationTextA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproValidationTextS"}

Sets or returns a value that specifies the text of the message that your application displays if the value of a **Field** object doesn't satisfy the validation rule specified by the **ValidationRule** property setting (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **String** that specifies the text displayed if a user tries to enter an invalid value for a field. For an object not yet appended to a collection, this property is read/write. For a **Recordset** object, this property setting is read-only. For a **TableDef** object, this property setting is read-only for a linked table and read/write for a base table.

Remarks

For a **Field** object, use of the **ValidationText** property depends on the object that contains the **Fields** collection to which the **Field** object is appended, as the following table shows.

Object appended to	Usage
Index	Not supported
QueryDef	Read-only
Recordset	Read-only
Relation	Not supported
TableDef	Read/write

Value Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproValueC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproValueA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproValueS"}

Sets or returns the value of an object.

Settings and Return Values

The setting or return value is a **Variant** data type that evaluates to a value appropriate for the data type, as specified by the **Type** property of an object.

Remarks

Generally, the **Value** property is used to retrieve and alter data in **Recordset** objects.

The **Value** property is the default property of the **Field**, **Parameter**, and **Property** objects. Therefore, you can set or return the value of one of these objects by referring to them directly instead of specifying the **Value** property.

Trying to set or return the **Value** property in an inappropriate context (for example, the **Value** property of a **Field** object in the **Fields** collection of a **TableDef** object) will cause a trappable error.

Notes

- In an ODBCDirect workspace, you cannot read or set the **Value** property of a **Recordset** field more than once without refreshing the current record. For example, to read and then set the **Value** property, first read the property, then use the **Move** 0 method to refresh the current record, then write the new value.
- When reading decimal values from a Microsoft SQL Server database, they will be formatted using scientific notation through a Microsoft Jet workspace, but will appear as normal decimal values through an ODBCDirect workspace.

Version Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproVersionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproVersionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproVersionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproVersionS"}

- Microsoft Jet workspace – On the **DBEngine** object, returns the version of DAO currently in use. On the **Database** object, returns the version of Jet that created the .mdb file.
- ODBCDirect workspace – On the **DBEngine** object, returns the version of DAO currently in use. On the **Database** object, returns the version of the ODBC driver currently in use.

Return Values

The return value is a **String** that evaluates to a version number, formatted as follows.

- Microsoft Jet workspace – represents the version number in the form "*major.minor*". For example, "3.0". The product version number consists of the version number (3), a period, and the release number (0).
- ODBCDirect workspace – represents the DAO version number in the form "*major.minor*", or represents the ODBC driver version number in the form "*major.minor.build*". For example, the **DBEngine.Version** value of "3.5" indicated DAO version 3.5. A **Database** object's **Version** value of 2.50.1032 indicates that the current instance of DAO is connected to ODBC version 2.5, build 1032.

Remarks

In a Microsoft Jet workspace, the **Version** property of a **Database** object corresponds to a version of the Microsoft Jet database engine, and doesn't necessarily match the version number of the Microsoft product with which the database engine was included. For example, the **Version** property of a **Database** object created with Microsoft Visual Basic 3.0 will be 1.1, not 3.0.

The following table shows which version of the database engine was included with various versions of Microsoft products.

Microsoft Jet Version (year released)	Microsoft Access	Microsoft Visual Basic	Microsoft Excel	Microsoft Visual C++
1.0 (1992)	1.0	N/A	N/A	N/A
1.1 (1993)	1.1	3.0	N/A	N/A
2.0 (1994)	2.0	N/A	N/A	N/A
2.5 (1995)	N/A	4.0 (16-bit)	N/A	N/A
3.0 (1995)	‘95 (7.0)	4.0 (32-bit)	‘95 (7.0)	4.x
3.5 (1996)	‘97 (8.0)	5.0	‘97 (8.0)	5.0

DesignMasterID Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDesignMasterIDC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDesignMasterIDX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDesignMasterIDA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDesignMasterIDS"}

Sets or returns a 16-byte value that uniquely identifies the Design Master in a replica set (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **GUID** that uniquely identifies the Design Master.

Remarks

You should set the **DesignMasterID** property only if you need to move the current Design Master. Setting this property makes a specific replica in the replica set the Design Master.

Caution Never create a second Design Master in a replica set. The existence of a second Design Master can result in the loss of data.

Under extreme circumstances — for example, if the Design Master is erased or corrupted — you can set this property at the current replica. However, setting this property at a replica when there is already another Design Master in the set might partition your replica set into two irreconcilable sets and prevent any further synchronization of data.

If you decide to make a replica the new Design Master for the set, synchronize it with all the replicas in the replica set before setting the **DesignMasterID** property in the replica. The replica must be open in exclusive mode in order to make it the Design Master.

If you make a replica that is designated read-only into the Design Master, the target replica is made read/write; the old Design Master also remains read/write.

The **DesignMasterID** property setting is stored in the MSysRepInfo system table.

KeepLocal Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproKeepLocalC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproKeepLocalX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproKeepLocalA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproKeepLocalS"}

Sets or returns a value on a table, query, form, report, macro, or module that you do not want to replicate when the database is replicated (Microsoft Jet workspaces only).

Note Before getting or setting the **KeepLocal** property on a **TableDef**, or **QueryDef** object, you must create it by using the **CreateProperty** method and append it to the **Properties** collection for the object.

Settings and Return Values

The setting or return value is a **Text** data type. If you set this property to `"T",` the object will remain local when the database is replicated. You can't use the **KeepLocal** property on objects after they have been replicated.

Remarks

Once you set the **KeepLocal** property, it will appear in the **Properties** collection for the **Document** object representing the host object.

Before setting the **KeepLocal** property, you should check the value of the **Replicable** property.

After you make a database replicable, all new objects created within the Design Master, or in any other replicas in the set, are local objects. Local objects remain in the replica in which they're created and aren't copied throughout the replica set. Each time you make a new replica in the set, the new replica contains all the replicable objects from the source replica, but none of the local objects from the source replica.

If you create a new object in a replica and want to change it from local to replicable so that all users can use it, you can either create the object in or import it into the Design Master. Be sure to delete the local object from any replicas; otherwise, you will encounter a design error. After the object is part of the Design Master, set the object's **Replicable** property to **True**.

The object on which you are setting the **KeepLocal** property might have already inherited that property from another object. However, the value set by the other object has no effect on the behavior of the object you want to keep local. You must explicitly set the property for each object.

Replicable Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproReplicableC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproReplicableX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproReplicableA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproReplicableS"}

Sets or returns a value that determines whether a database or object in a database can be replicated (Microsoft Jet workspaces only).

Note Before getting or setting the **Replicable** property on a **Database**, **TableDef**, or **QueryDef** object, you must create it by using the **CreateProperty** method and append it to the **Properties** collection for the object.

Setting and Return Values

The setting or return value is a **Text** data type.

On a **Database** object, setting this property to `"T"` makes the database replicable. Once you set the property to `"T"`, you can't change it; setting the property to `"F"` (or any value other than `"T"`) causes an error.

On an object in a database, setting this property to `"T"` replicates the object (and subsequent changes to the object) at all replicas in the replica set. You can also set this property in the object's property sheet in Microsoft Access.

Note Microsoft Jet 3.5 also supports the **Boolean** **ReplicableBool** property. Its functionality is identical to the **Replicable** property, except that it takes a **Boolean** value. Setting **ReplicableBool** to **True** makes the object replicable.

Remarks

Before setting the **Replicable** property on a database, make a backup copy of the database. If setting the **Replicable** property fails, you should delete the partially replicated database, make a new copy from the backup, and try again.

When you set this property on a **Database** object, Microsoft Jet adds fields, tables, and properties to objects within the database. Microsoft Jet uses these fields, tables, and properties to synchronize database objects. For example, all existing tables have three new fields added to them that help identify which records have changed. The addition of these fields and other objects increase the size of your database.

On forms, reports, macros, and modules defined by a host application (such as Microsoft Access), you set this property on the host-defined object through the host user interface. Once set, the **Replicable** property will appear in the **Properties** collection for the **Document** object representing the host object.

If the **Replicable** property has already been set on an object using the **Replicated** check box in the property sheet for the object, you cannot set the **Replicable** property in code.

When you create a new table, query, form, report, macro, or module at a replica, the object is considered local and is stored only at that replica. If you want users at other replicas to be able to use the object, you must change it from local to replicable. Either create the object at or import it into the Design Master and then set the **Replicable** property to `"T"`.

The object on which you are setting the **Replicable** property might have already inherited that property from another object. However, the value set by the other object has no effect on the behavior of the object you want to make replicable. You must explicitly set the property for each object.

ReplicaID Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproReplicaIDC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproReplicaIDX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproReplicaIDA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproReplicaIDS"}

Returns a 16-byte value that uniquely identifies a database replica (Microsoft Jet workspaces only).

Return Values

The return value is a **GUID** value that uniquely identifies the replica or Design Master.

Remarks

The Microsoft Jet database engine automatically generates this value when you create a new replica.

The **ReplicaID** property of each replica (and the Design Master) is stored in the MSysReplicas system table.

SystemDB Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproSystemDBC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproSystemDBX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproSystemDBA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproSystemDBS"}

Sets or returns the path for the current location of the workgroup information file (Microsoft Jet workspaces only).

Settings and Return Values

The setting or return value is a **String** describing the fully resolved path to the workgroup information file.

Remarks

The Microsoft Jet database engine allows you to define a workgroup and set different access permissions to each object in the database for each user in the workgroup. The workgroup is defined by the workgroup information file, typically called "system.mda". For users to gain access to the secured objects in your database, DAO must have the location of this workgroup information file. The location can be identified to DAO either by specifying it in the Windows Registry or by setting the **SystemDB** property. On setup, the default setting is simply "system.mda" with no path.

For this option to have any effect, you must set the **SystemDB** property before your application initializes the **DBEngine** object (that is, before creating an instance of any other DAO object). The scope of this setting is limited to your application and can't be changed without restarting your application.

PartialReplica Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproPartialReplicaC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproPartialReplicaX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproPartialReplicaA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproPartialReplicaS"}

Sets or returns a value on a **Relation** object indicating whether that relation should be considered when populating a partial replica from a full replica. (Microsoft Jet databases only.)

Settings and Return Values

The setting or return value is a **Boolean** data type that is **True** when the relation should be enforced during synchronization.

Remarks

This property enables you to replicate data from the full replica to the partial replica based on relationships between tables. You can use the **PartialReplica** property when setting the **ReplicaFilter** property alone can't adequately specify what data should be replicated to the partial. For example, suppose you have a database in which the Customers table has a one-to-many relationship with the Orders table, and you want to configure a partial replica that only replicates orders from customers in the California region (instead of all orders). It is not possible to set the **ReplicaFilter** property on the Orders table to `Region = 'CA'` because the Region field is in the Customers table, not the Orders table.

To replicate all orders from the California region, you must indicate that the relation between the Orders and Customers tables will be active during replication. Once you've created a partial replica, the following steps will populate it with all orders from the California region:

1. Set the **ReplicaFilter** property on the Customers **TableDef** object to `"Region = 'CA'"`.
2. Set the value of the **PartialReplica** property to **True** on the **Relation** object corresponding to the relationship between Orders and Customers.
3. Invoke the **PopulatePartial** method.

Caution When you set a replica filter or replica relation, be aware that records in the partial replica that don't satisfy the restriction criteria will be removed from the partial replica, but not from the full replica. For example, suppose you set the **ReplicaFilter** property on the Customers **TableDef** in the partial replica to `"Region = 'CA'"` and you then repopulate the database. This will insert or update all records for California-based customers. If you then reset the **ReplicaFilter** property to `"Region = 'FL'"` and repopulate the database, all California region records in the partial replica will be removed, and all records from Florida-based customers will be inserted from the full replica. No records in the full replica will be deleted.

Before setting either the **ReplicaFilter** or **PartialReplica** property, it's a good idea to synchronize the partial replica in which you are setting these properties with the full replica. This will ensure that pending changes in the partial replica will be merged into the full replica before any records are removed in the partial replica.

ReplicaFilter Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproReplicaFilterC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproReplicaFilterX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproReplicaFilterA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproReplicaFilterS"}

Sets or returns a value on a **TableDef** object within a partial replica that indicates which subset of records is replicated to that table from a full replica. (Microsoft Jet databases only.)

Settings And Return Values

The setting or return value is a **String** or **Boolean** that indicates which subset of records is replicated, as specified in the following table:

Value	Description
A string	A criteria that a record in the partial replica table must satisfy in order to be replicated from the full replica.
True	Replicates all records.
False	(Default) Doesn't replicate any records.

Remarks

This property is similar to an SQL WHERE clause (without the word WHERE), but you cannot specify subqueries, aggregate functions (such as **Count**), or user-defined functions within the criteria.

You can only synchronize data between a full replica and a partial replica. You can't synchronize data between two partial replicas. Also, with partial replication you can set restrictions on which records are replicated, but you can't indicate which fields are replicated.

Usually, you reset a replica filter when you want to replicate a different set of records. For example, when a sales representative temporarily takes over another sales representative's region, the database application can temporarily replicate data for both regions and then return to the previous filter. In this scenario, the application resets the **ReplicaFilter** property and then repopulates the partial replica.

If your application changes replica filters, you should follow these steps:

1. Use the **Synchronize** method to synchronize your full replica with the partial replica in which the filters are being changed.
2. Use the **ReplicaFilter** property to make the desired changes to the replica filter.
3. Use the **PopulatePartial** method to remove all records from the partial replica and transfer all records from the full replica that meet the new replica filter criteria.

To remove a filter, set the **ReplicaFilter** property to **False**. If you remove all filters and invoke the **PopulatePartial** method, no records will appear in any replicated tables in the partial replica.

Note If a replica filter has changed, and the **Synchronize** method is invoked without first invoking **PopulatePartial**, a trappable error occurs.

BatchCollisionCount Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproBatchCollisionCountC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproBatchCollisionCountX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproBatchCollisionCountA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproBatchCollisionCountS"}

Returns the number of records that did not complete the last batch update (ODBCDirect workspaces only).

Return Value

The return value is a **Long** that indicates the number of failing records, or 0 if all records were successfully updated.

Remarks

This property indicates how many records encountered collisions or otherwise failed to update during the last batch update attempt. The value of this property corresponds to the number of bookmarks in the **BatchCollisions** property.

If you set the working **Recordset** object's **Bookmark** property to bookmark values in the **BatchCollisions** array, you can move to each record that failed to complete the most recent batch **Update** operation.

After the collision records are corrected, a batch-mode **Update** method can be called again. At this point DAO attempts another batch update, and the **BatchCollisions** property again reflects the set of records that failed the second attempt. Any records that succeeded in the previous attempt are not sent in the current attempt, because they now have a **RecordStatus** property of **dbRecordUnmodified**. This process can continue as long as collisions occur, or until you abandon the updates and close the result set.

BatchCollisions Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproBatchCollisionsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproBatchCollisionsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproBatchCollisionsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproBatchCollisionsS"}

Returns an array of bookmarks indicating the rows that generated collisions in the last batch update operation (ODBCDirect workspaces only).

Return Value

The return value is a variant expression containing an array of bookmarks.

Remarks

This property contains an array of bookmarks to rows that encountered a collision during the last attempted batch **Update** call. The **BatchCollisionCount** property indicates the number of elements in the array.

If you set the working **Recordset** object's **Bookmark** property to bookmark values in the **BatchCollisions** array, you can move to each record that failed to complete the most recent batch-mode **Update** operation.

After the collision records are corrected, you can call the batch mode **Update** method again. At this point DAO attempts another batch update, and the **BatchCollisions** property again reflects the set of records that failed the second attempt. Any records that succeeded in the previous attempt are not sent in the current attempt, as they now have a **RecordStatus** property of **dbRecordUnmodified**. This process can continue as long as collisions occur, or until you abandon the updates and close the result set.

This array is re-created each time you execute a batch-mode **Update** method.

BatchSize Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproBatchSizeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproBatchSizeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproBatchSizeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproBatchSizeS"}

Sets or returns the number of statements sent back to the server in each batch (ODBCDirect workspaces only).

Settings And Return Values

The setting or return value is a **Long** that indicates the number of batched statements sent the server in a single batch update. The default value is 15.

Remarks

The **BatchSize** property determines the batch size used when sending statements to the server in a batch update. The value of the property determines the number of statements sent to the server in one command buffer. By default, 15 statements are sent to the server in each batch. This property can be changed at any time. If a database server doesn't support statement batching, you can set this property to 1, causing each statement to be sent separately.

CacheStart Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproCacheStartC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproCacheStartX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproCacheStartA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproCacheStartS"}

Sets or returns a value that specifies the bookmark of the first record in a dynaset-type **Recordset** object containing data to be locally cached from an ODBC data source (Microsoft Jet workspaces only).

Settings And Return Values

The setting or return value is a **String** that specifies a bookmark.

Remarks

Data caching improves the performance of an application that retrieves data from a remote server through dynaset-type **Recordset** objects. A cache is a space in local memory that holds the data most recently retrieved from the server in the event that the data will be requested again while the application is running. When data is requested, the Microsoft Jet database engine checks the cache for the requested data first rather than retrieving it from the server, which takes more time. Only data from an ODBC data source can be saved in the cache.

Any Microsoft Jet-connected ODBC data source, such as a linked table, can have a local cache. To create the cache, open a **Recordset** object from the remote data source, set the **CacheSize** and **CacheStart** properties, and then use the **FillCache** method or step through the records using the Move methods.

The **CacheStart** property setting is the bookmark of the first record in the **Recordset** object to be cached. You can use the bookmark of any record to set the **CacheStart** property. Make the record you want to start the cache the current record, and set the **CacheStart** property equal to the **Bookmark** property.

The Microsoft Jet database engine requests records within the cache range from the cache, and it requests records outside the cache range from the server.

Records retrieved from the cache don't reflect changes made concurrently to the source data by other users.

To force an update of all the cached data, set the **CacheSize** property of the **Recordset** object to 0, set it to the size of the cache you originally requested, and then use the **FillCache** method.

Connection Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproConnectionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproConnectionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproConnectionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproConnectionS"}

On a **Database** object, returns the **Connection** object that corresponds to the database (ODBCDirect workspaces only).

On a **Recordset** object, returns the **Connection** object that owns the **Recordset** (ODBCDirect workspaces only).

Settings And Return Values

The return value is an object variable that represents the **Connection**. On a **Database** object, the **Connection** property is read-only, while on a **Recordset** object the property is read-write.

Remarks

On a **Database** object, use the **Connection** property to obtain a reference to a **Connection** object that corresponds to the **Database**. In DAO, a **Connection** object and its corresponding **Database** object are simply two different object variable references to the same object. The **Database** property of a **Connection** object and the **Connection** property of a **Database** object make it easier to change connections to an ODBC data source through the Microsoft Jet database engine to use ODBCDirect.

Database Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDatabaseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDatabaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDatabaseA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDatabaseS"}

Returns the **Database** object that corresponds to this connection (ODBCDirect workspaces only).

Return Values

The return value is an object variable that represents a **Database** object.

Remarks

On a **Connection** object, use the **Database** property to obtain a reference to a **Database** object that corresponds to the **Connection**. In DAO, a **Connection** object and its corresponding **Database** object are simply two different object variable references to the same object. The **Database** property of a **Connection** object and the **Connection** property of a **Database** object make it easier to change connections to an ODBC data source through the Microsoft Jet database engine to use ODBCDirect.

DefaultCursorDriver Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDefaultCursorDriverC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDefaultCursorDriverX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDefaultCursorDriverA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDefaultCursorDriverS"}

Sets or returns the type of cursor driver used on the connection created by the **OpenConnection** or **OpenDatabase** methods (ODBCDirect workspaces only).

Settings And Return Values

The setting or return value is a **Long** that can be set to one of the following constants:

Constant	Description
dbUseDefaultCursor	(Default) Uses server-side cursors if the server supports them; otherwise use the ODBC Cursor Library.
dbUseODBCCursor	Always uses the ODBC Cursor Library. This option provides better performance for small result sets, but degrades quickly for larger result sets.
dbUseServerCursor	Always uses server-side cursors. For most large operations this option provides better performance, but might cause more network traffic.
dbUseClientBatchCursor	Always uses the client batch cursor library. This option is required for batch updates.
dbUseNoCursor	Opens all cursors (that is, **Recordset** objects) as forward-only type, read-only, with a rowset size of 1. Also known as "cursorless queries."

Remarks

This property setting only affects connections established after the property has been set. Changing the **DefaultCursorDriver** property has no effect on existing connections.

DefaultType Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDefaultTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDefaultTypeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDefaultTypeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDefaultTypeS"}

Sets or returns a value that indicates what type of workspace (Microsoft Jet or ODBCDirect) will be used by the next **Workspace** object created.

Settings And Return Values

The setting or return value is a **Long** that can be set to either of the following constants:

Constant	Description
dbUseJet	Creates **Workspace** objects connected to the Microsoft Jet database engine
dbUseODBC	Creates **Workspace** objects connected to an ODBC data source

Remarks

The setting can be overridden for a single **Workspace** by setting the *type* argument to the **CreateWorkspace** method.

Direction Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproDirectionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproDirectionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproDirectionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproDirectionS"}

Sets or returns a value that indicates whether a **Parameter** object represents an input parameter, an output parameter, both, or the return value from the procedure (ODBCDirect workspaces only).

Settings And Return Values

The setting or return value is a **Long** that can be set to one of the following constants:

Constant	Description
dbParamInput	(Default) Passes information to the procedure.
dbParamInputOutput	Passes information both to and from the procedure.
dbParamOutput	Returns information from the procedure as in an output parameter in SQL.
dbParamReturnValue	Passes the return value from a procedure.

Remarks

Use the **Direction** property to determine whether the parameter is an input parameter, output parameter, both, or the return value from the procedure. Some ODBC drivers do not provide information on the direction of parameters to a SELECT statement or procedure call. In these cases, it is necessary to set the direction prior to executing the query.

For example, the following procedure returns a value from a stored procedure named "get_employees":

```
{? = call get_employees}
```

This call produces one parameter — the return value. You need to set the direction of this parameter to **dbParamOutput** or **dbParamReturnValue** before executing the **QueryDef**.

You need to set all parameter directions except **dbParamInput** before accessing or setting the values of the parameters and before executing the **QueryDef**.

You should use **dbParamReturnValue** for return values, but in cases where that option is not supported by the driver or the server, you can use **dbParamOutput** instead.

Note The Microsoft SQL Server 6.0 driver automatically sets the **Direction** property for all procedure parameters. Not all ODBC drivers can determine the direction of a query parameter. In these cases, it is necessary to set the direction prior to executing the query.

FieldSize Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproFieldSizeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproFieldSizeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproFieldSizeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproFieldSizeS"}

Returns the number of bytes used in the database (rather than in memory) of a Memo or Long Binary **Field** object in the **Fields** collection of a **Recordset** object.

Return Values

The return value is a **Long** that indicates the number of characters (for a Memo field) or the number of bytes (for a Long Binary field).

Remarks

You can use **FieldSize** with the **AppendChunk** and **GetChunk** methods to manipulate large fields.

Because the size of a Long Binary or Memo field can exceed 64K, you should assign the value returned by **FieldSize** to a variable large enough to store a **Long** variable.

To determine the size of a **Field** object other than Memo and Long Binary types, use the **Size** property.

Note In an ODBCDirect workspace, the **FieldSize** property is not available in the following situations:

- If the database server or ODBC driver does not support server-side cursors.
- If you are using the ODBC cursor library (that is, the **DefaultCursorDriver** property is set to **dbUseODBC**, or to **dbUseDefault** when the server does not support server-side cursors).
- If you are using a cursorless query (that is, the **DefaultCursorDriver** property is set to **dbUseNoCursor**).

For example, Microsoft SQL Server version 4.21 does not support server-side cursors, so the **FieldSize** property is not available.

MaxRecords Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproMaxRecordsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproMaxRecordsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproMaxRecordsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproMaxRecordsS"}

Sets or returns the maximum number of records to return from a query.

Settings And Return Values

The setting or return value is a **Long** that represents the number of records to be returned. The default value is 0, indicating no limit on the number of records returned.

Remarks

Once the number of rows specified by **MaxRecords** is returned to your application in a **Recordset**, the query processor will stop returning additional records even if more records would qualify for inclusion in the **Recordset**. This property is useful in situations where limited client resources prohibit management of large numbers of records.

OriginalValue Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproOriginalValueC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproOriginalValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproOriginalValueA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproOriginalValueS"}

Returns the value of a **Field** in the database that existed when the last batch update began (ODBCDirect workspaces only).

Return Values

The return value is a variant expression.

Remarks

During an optimistic batch update, a collision may occur where a second client modifies the same field and record in between the time the first client retrieves the data and the first client's update attempt. The **OriginalValue** property contains the value of the field at the time the last batch **Update** began. If this value does not match the value actually in the database when the batch **Update** attempts to write to the database, a collision occurs. When this happens, the new value in the database will be accessible through the **VisibleValue** property.

Prepare Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproPrepareC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproPrepareX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproPrepareA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproPrepareS"}

Sets or returns a value that indicates whether the query should be prepared on the server as a temporary stored procedure, using the ODBC **SQLPrepare** API function, prior to execution, or just executed using the ODBC **SQLExecDirect** API function (ODBCDirect workspaces only).

Settings and Return Values

The setting or return value is a **Long** value that can be one of the following constants:

Constant	Description
dbQPrepare	(Default) The statement is prepared (that is, the ODBC **SQLPrepare** API is called).
dbQUnprepare	The statement is not prepared (that is, the ODBC **SQLExecDirect** API is called).

Remarks

You can use the **Prepare** property to either have the server create a temporary stored procedure from your query and then execute it, or just have the query executed directly. By default the **Prepare** property is set to **dbQPrepare**. However, you can set this property to **dbQUnprepare** to prohibit preparing of the query. In this case, the query is executed using the **SQLExecDirect** API.

Creating a stored procedure can slow down the initial operation, but increases performance of all subsequent references to the query. However, some queries cannot be executed in the form of stored procedures. In these cases, you must set the **Prepare** property to **dbQUnprepare**.

If **Prepare** is set to **dbQPrepare**, this can be overridden when the query is executed by setting the **Execute** method's *options* argument to **dbExecDirect**.

Note The ODBC **SQLPrepare** API is called as soon as the DAO **SQL** property is set. Therefore, if you want to improve performance using the **dbQUnprepare** option, you must set the **Prepare** property before setting the **SQL** property.

RecordStatus Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproRecordStatusC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproRecordStatusX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproRecordStatusA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproRecordStatusS"}

Returns a value indicating the update status of the current record if it is part of a batch update (ODBCDirect workspaces only).

Return Values

The return value is a **Long** that can be any of the following constants:

Constant	Description
dbRecordUnmodified	(Default) The record has not been modified or has been updated successfully.
dbRecordModified	The record has been modified and not updated in the database.
dbRecordNew	The record has been inserted with the **AddNew** method, but not yet inserted into the database.
dbRecordDeleted	The record has been deleted, but not yet deleted in the database.
dbRecordDBDeleted	The record has been deleted locally *and* in the database.

Remarks

The value of the **RecordStatus** property indicates whether and how the current record will be involved in the next optimistic batch update.

When a user changes a record, the **RecordStatus** for that record automatically changes to **dbRecordModified**. Similarly, if a record is added or deleted, **RecordStatus** reflects the appropriate constant. When you then use a batch-mode **Update** method, DAO will submit an appropriate operation to the remote server for each record, based on the record's **RecordStatus** property.

StillExecuting Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproStillExecutingC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproStillExecutingX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproStillExecutingA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproStillExecutingS"}

Indicates whether or not an asynchronous operation (that is, a method called with the **dbRunAsync** option) has finished executing (ODBCDirect workspaces only).

Settings And Return Values

The return value is a **Boolean** that is **True** if the query is still executing, and **False** if the query has completed.

Remarks

Use the **StillExecuting** property to determine if the most recently called asynchronous **Execute**, **MoveLast**, **OpenConnection,** or **OpenRecordset** method (that is, a method executed with the **dbRunAsync** option) is complete. While the **StillExecuting** property is **True**, any returned object cannot be accessed.

The following table shows what method is evaluated when you use **StillExecuting** on a particular type of object.

If StillExecuting is used on	This asynchronous method is evaluated
Connection	**Execute** or **OpenConnection**
QueryDef	**Execute**
Recordset	**MoveLast** or **OpenRecordset**

Once the **StillExecuting** property on a **Connection** or **Recordset** object returns **False**, follwing the **OpenConnection** or **OpenRecordset** call that returns the associated **Recordset** or **Connection** object, the object can be referenced. So long as **StillExecuting** remains **True**, the object may not be referenced, other than to read the **StillExecuting** property. When you use the **NextRecordset** method to complete processing of a **Recordset**, the **StillExecuting** property is reset to **True** while subsequent result sets are retrieved.

Use the **Cancel** method to terminate execution of a task in progress.

UpdateOptions Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproUpdateOptionsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproUpdateOptionsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproUpdateOptionsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproUpdateOptionsS"}

Sets or returns a value that indicates how the WHERE clause is constructed for each record during a batch update, and whether the batch update should use an UPDATE statement or a DELETE followed by an INSERT (ODBCDirect workspaces only).

Settings And Return Values

The setting or return value is a **Long** that can be any of the following constants:

Constant	Description
dbCriteriaKey	(Default) Uses just the key column(s) in the where clause.
dbCriteriaModValues	Uses the key column(s) and all updated columns in the where clause.
dbCriteriaAllCols	Uses the key column(s) and all the columns in the where clause.
dbCriteriaTimeStamp	Uses just the timestamp column if available (will generate a run-time error if no timestamp column is in the result set).
dbCriteriaDeleteInsert	Uses a set of DELETE and INSERT statements for each modified row.
dbCriteriaUpdate	(Default) Uses an UPDATE statement for each modified row.

Remarks

When a batch-mode **Update** is executed, DAO and the client batch cursor library create a series of SQL UPDATE statements to make the needed changes. An SQL WHERE clause is created for each update to isolate the records that are marked as changed by the **RecordStatus** property. Because some remote servers use triggers or other ways to enforce referential integrity, is it often important to limit the fields being updated to just those affected by the change. To do this, set the **UpdateOptions** property to one of the constants **dbCriteriaKey**, **dbCriteriaModValues**, **dbCriteriaAllCols**, or **dbCriteriaTimeStamp**. This way, only the absolute minimum amount of trigger code is executed. As a result, the update operation is executed more quickly, and with fewer potential errors.

You can also concatenate either of the constants **dbCriteriaDeleteInsert** or **dbCriteriaUpdate** to determine whether to use a set of SQL DELETE and INSERT statements or an SQL UPDATE statement for each update when sending batched modifications back to the server. In the former case, two separate operations are required to update the record. In some cases, especially where the remote system implements DELETE, INSERT, and UPDATE triggers, choosing the correct **UpdateOptions** property setting can significantly impact performance.

If you don't specify any constants, **dbCriteriaUpdate** and **dbCriteriaKey** will be used.

Newly added records will always generate INSERT statements and deleted records will always generate DELETE statements, so this property only applies to how the cursor library updates modified records.

VisibleValue Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"daproVisibleValueC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"daproVisibleValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"daproVisibleValueA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"daproVisibleValueS"}

Returns a value currently in the database that is newer than the **OriginalValue** property as determined by a batch update conflict (ODBCDirect workspaces only).

Return Values

The return value is a variant expression.

Remarks

This property contains the value of the field that is currently in the database on the server. During an optimistic batch update, a collision may occur where a second client modified the same field and record in between the time the first client retrieved the data and the first client's update attempt. When this happens, the value that the second client set will be accessible through this property.

Data Access Objects (DAO) Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"damscDataAccessConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"damscDataAccessConstantsX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"damscDataAccessConstantsS"}

Data Access Objects (DAO) provides built-in constants that you can use with methods or properties. These constants all begin with the letters **db** and are documented with the method or property to which they apply.

Legend:

Read-only

Read/write

AllPermissions Property Constants (All Are)

For any **Container** or **Document** object:

Constant	Description
dbSecReadDef	Allows user to read the table definition, including column and index information.
dbSecWriteDef	Allows user to modify or delete the table definition, including column and index information.
dbSecRetrieveData	Allows user to retrieve data from the **Document** object.
dbSecInsertData	Allows user to add records.
dbSecReplaceData	Allows user to modify records.
dbSecDeleteData	Allows user to delete records.

The **Databases** container or any **Document** object in a **Documents** collection may include the following:

Constant	Description
dbSecDeleteData	Allows user to delete records.
dbSecDBAdmin	Allows user to replicate the database and change the database password.
dbSecDBCreate	Allows user to create new databases. This setting is valid only on the Databases container in the workgroup information file (System.mdw).
dbSecDBExclusive	Allows user exclusive access to the database.
dbSecDBOpen	Allows user to open the database.

Attributes Property Constants

For any **Field** object, the **Attributes** property may include the following:

Constant	Description
dbFixedField	Fixed field size

	(default for Numeric fields)
dbVariableField	Variable field size (Text fields only)
dbAutoIncrField	New record field value incremented to unique **Long** integer (in a Microsoft Jet workspace, available only on **TableDef** objects opened from .mdb files)
dbUpdatableField	Field is updatable
dbDescending	Field sorted in descending order (Microsoft Jet workspaces only)
dbHyperlinkField	The field contains hyperlink information (Memo fields in Microsoft Jet workspaces only)
dbSystemField	The field is a replication field (on a **TableDef** object in Microsoft Jet databases only)

For any **Relation** object, the **Attributes** property may include the following:

Constant	Description
dbRelationUnique	One-to-one relationship
dbRelationDontEnforce	Relationship not enforced (no referential integrity)
dbRelationInherited	Relationship exists in the database containing the two linked tables
dbRelationUpdateCascade	Updates cascade
dbRelationDeleteCascade	Deletions cascade

For any **TableDef** object, the **Attributes** property may include the following:

Constant	Description
dbAttachExclusive	Opens a linked Microsoft Jet database engine table for exclusive use.
dbAttachSavePWD	Saves user ID and password for linked remote table.
dbSystemObject	System table
dbHiddenObject	Hidden table (for temporary use)

dbAttachedTable	Linked non-ODBC database table
dbAttachedODBC	Linked ODBC database table

CollatingOrder Property Constants (All Are)

Constant	Description
dbSortArabic	Arabic collating order
dbSortChineseSimplified	Simplified Chinese collating order
dbSortChineseTraditional	Traditional Chinese collating order
dbSortCyrillic	Russian collating order
dbSortCzech	Czech collating order
dbSortDutch	Dutch collating order
dbSortGeneral	English, German, French, and Portuguese collating order
dbSortGreek	Greek collating order
dbSortHebrew	Hebrew collating order
dbSortHungarian	Hungarian collating order
dbSortIcelandic	Icelandic collating order
dbSortJapanese	Japanese collating order
dbSortKorean	Korean collating order
dbSortNeutral	Neutral collating order
dbSortNorw	Norwegian and Danish collating order
dbSortPDXIntl	Paradox international collating order
dbSortPDXNor	Paradox Norwegian and Danish collating order
dbSortPDXSwe	Paradox Swedish and Finnish collating order
dbSortPolish	Polish collating order
dbSortSlovenian	Slovenian collating order
dbSortSpanish	Spanish collating order
dbSortSwedFin	Swedish and Finnish collating order
dbSortThai	Thai collating order
dbSortTurkish	Turkish collating order
dbSortUndefined	Collating order undefined or unknown

DefaultCursorDriver Property (All are)

Constant	Description
dbUseDefaultCursor	(Default) Uses server-side cursors if the server supports them; otherwise uses the ODBC Cursor Library.
dbUseODBCCursor	Always uses the ODBC Cursor Library. This option provides better performance for small result sets, but degrades quickly for larger result

	sets.
dbUseServerCursor	Always uses server-side cursors. For most large operations this option provides better performance, but might cause more network traffic.
dbUseClientBatchCursor	Always uses the FoxPro Cursor Library. This option is required for performing batch updates.
dbUseNoCursor	Opens all cursors (that is, **Recordset** objects) as forward-only type, read-only, with a rowset size of 1. Also known as "cursorless queries."

Direction Property Constants (All Are)

Constant	Description
dbParamInput	(Default) Passes information to the procedure.
dbParamInputOutput	Passes information both to and from the procedure.
dbParamOutput	Returns information from the procedure as in an output parameter in SQL.
dbParamReturnValue	Passes the return value from a procedure.

EditMode Property Constants (All Are)

Constant	Description
dbEditNone	No editing operation in effect.
dbEditInProgress	**Edit** method invoked.
dbEditAdd	**AddNew** method invoked.

Permissions Property Constants (All are)

For any **Container** object, the **Permissions** property may include the following:

Constant	Description
dbSecNoAccess	Denies user access to the object.
dbSecFullAccess	Allows user full access to the object.
dbSecDelete	Allows user to delete the object.
dbSecReadSec	Allows user to read the object's security-related information.
dbSecWriteSec	Allows user to alter access permissions.
dbSecWriteOwner	Allows user to change the **Owner** property setting.

For any database **Container**, the **Permissions** property may include any of the following (All are

):

Constant	Description
dbSecDBAdmin	Gives user permission to make a database replicable and change the database password.
dbSecDBCreate	Allows user to create new databases (valid only on the databases **Container** object in the system database).
dbSecDBOpen	Allows user to open the database.
dbSecDBExclusive	Allows user exclusive access.

For any tables **Container**, the **Permissions** property may include any of the following (All are

):

Constant	Description
dbSecCreate	Allows user to create new tables (valid only with a **Container** object that represents a table or with the databases **Container** object in the system database).
dbSecReadDef	Allows user to read the table definition, including column and index information.
dbSecWriteDef	Allows user to modify or delete the table definition, including column and index information.
dbSecRetrieveData	Allows user to retrieve data from the document.
dbSecInsertData	Allows user to add records.
dbSecReplaceData	Allows user to modify records.
dbSecDeleteData	Allows user to delete records.

For any **Document** object, the **Permissions** property may include any of the following (All are

):

Constant	Description
dbSecCreate	Allows user to create new tables (valid only with a **Container** object that represents a table).
dbSecDBCreate	Allows user to create new databases (valid only on the databases **Container** object in the system database).
dbSecDBOpen	Allows user to open the database.
dbSecDBExclusive	Allows user exclusive access.
dbSecDelete	Allows user to delete the object.
dbSecDeleteData	Allows user to delete records.
dbSecFullAccess	Allows user full access to the object.
dbSecInsertData	Allows user to add records.
dbSecReadDef	Allows user to read the table definition, including column and index

	information.
dbSecReadSec	Allows user to read the object's security-related information.
dbSecReplaceData	Allows user to modify records.
dbSecRetrieveData	Allows user to retrieve data from the document.
dbSecWriteDef	Allows user to modify or delete the table definition, including column and index information.
dbSecWriteSec	Allows user to alter access permissions.
dbSecWriteOwner	Allows user to change the **Owner** property setting.

Prepare Property Constants (All Are)

Constant	Description
dbQPrepare	(Default) The statement is prepared (that is, the ODBC **SQLPrepare** API is called).
dbQUnprepare	The statement is not prepared (that is, the ODBC **SQLExecDirect** API is called).

RecordStatus Property Constants (All Are)

Constant	Description
dbDBDeleted	The record has been deleted locally and in the database.
dbDeleted	The record has been deleted, but not yet deleted in the database.
dbRecordModified	The record has been modified and not updated in the database.
dbRecordNew	The record has been inserted with the AddNew method, but not yet inserted into the database.
dbRecordUnmodified	(Default) The record has not been modified or has been updated successfully.

Type Property Constants

For any **Field**, **Parameter**, or **Property** object, the **Type** property may include any of the follwing (All are):

Constant	Description
dbBigInt	Big Integer data (ODBCDirect only)
dbBinary	Binary data
dbBoolean	Boolean (True/False) data
dbByte	Byte (8-bit) data

dbChar	Character data (ODBCDirect only)
dbCurrency	Currency data
dbDate	Date value data
dbDecimal	Decimal data (ODBCDirect only)
dbDouble	Double-precision floating-point data
dbFloat	Floating-point data (ODBCDirect only)
dbGUID	GUID data
dbInteger	Integer data
dbLong	Long Integer data
dbLongBinary	Binary data (bitmap)
dbMemo	Memo data (extended text)
dbNumeric	Numeric data (ODBCDirect only)
dbSingle	Single-precision floating-point data
dbText	Text data (variable width)
dbTime	Data in time format (ODBCDirect only)
dbTimeStamp	Data in time and date format (ODBCDirect only)
dbVarBinary	Variable Binary data (ODBCDirect only)

For any **QueryDef** object, the **Type** property may include any of the following (All are

):

Constant	Description
dbQAction	Action query
dbQAppend	Append query
dbQCompound	Compound query (ODBCDirect workspaces only)
dbQCrosstab	Crosstab query
dbQDDL	Data-definition language (DDL) query
dbQDelete	Delete query
dbQMakeTable	Make-table query
dbQProcedure	SQL procedure that executes a stored procedure (ODBCDirect workspaces only)
dbQSelect	Select query
dbQSetOperation	Set operation query
dbQSPTBulk	Bulk operation query
dbQSQLPassThrough	SQL pass-through query
dbQUpdate	Update query

For any **Recordset** object, the **Type** property may include any of the following (All are

):

Constants	Description
dbOpenDynamic	Opens a dynaset-type **Recordset**

	(ODBCDirect workspaces only)
dbOpenDynaset	Opens a dynaset-type **Recordset**
dbOpenForwardOnly	Opens a forward-only type **Recordset**
dbOpenSnapshot	Opens a snapshot-type **Recordset**
dbOpenTable	Opens a table-type **Recordset** (Microsoft Jet workspaces only)

UpdateOptions Property Constants (All Are)

Constant	Description
dbCriteriaKey	(Default) Uses just the key column(s) in the where clause.
dbCriteriaModValues	Uses the key column(s) and all updated columns in the where clause.
dbCriteriaAllCols	Uses the key column(s) and all the columns in the where clause.
dbCriteriaTimeStamp	Uses just the timestamp column if available (will generate a run-time error if no timestamp column is in the result set).
dbCriteriaDeleteInsert	Uses a pair of DELETE and INSERT statements for each modified row.
dbCriteriaUpdate	(Default) Uses an UPDATE statement for each modified row.

CompactDatabase, CreateDatabase Methods Locale Argument Constants (All Are)

Constant	Description
dbLangGeneral	English, German, French, Portuguese, Italian, and Modern Spanish
dbLangArabic	Arabic
dbLangChineseSimplified	Simplified Chinese
dbLangChineseTraditional	Traditional Chinese
dbLangCyrillic	Russian
dbLangCzech	Czech
dbLangDutch	Dutch
dbLangGreek	Greek
dbLangHebrew	Hebrew
dbLangHungarian	Hungarian
dbLangIcelandic	Icelandic
dbLangJapanese	Japanese
dbLangKorean	Korean
dbLangNordic	Nordic
dbLangNorwdan	Norwegian and Danish
dbLangPolish	Polish

dbLangSlovenian	Slovenian
dbLangSpanish	Spanish
dbLangSwedfin	Swedish and Finnish
dbLangThai	Thai
dbLangTurkish	Turkish

CompactDatabase Method Options Argument Constants (All Are)

Constant	Description
dbDecrypt	Decrypts database while compacting
dbEncrypt	Encrypts database
dbVersion10	Microsoft Jet database engine version 1.0
dbVersion11	Microsoft Jet database engine version 1.1
dbVersion20	Microsoft Jet database engine version 2.0
dbVersion30	Microsoft Jet database engine version 3.0

CreateDatabase Method Options Argument Constants (All Are)

Constant	Description
dbEncrypt	Encrypts database
dbVersion10	Microsoft Jet database engine version 1.0
dbVersion11	Microsoft Jet database engine version 1.1
dbVersion20	Microsoft Jet database engine version 2.0
dbVersion30	Microsoft Jet database engine version 3.0

CreateWorkspace Method Type Argument Constants

For any **Workspace** object **Type** property and **DBEngine** object **DefaultType** property, use any of the following: (All Are)

Constant	Description
dbUseODBC	The next workspace created will use ODBCDirect.
dbUseJet	The next workspace created will use the Microsoft Jet database engine.

Execute Method Options Argument Constants (All Are)

Constant	Description
dbDenyWrite	Denies write permission to other users (Microsoft Jet workspaces

	only).
dbInconsistent	Allows inconsistent updates (Microsoft Jet workspaces only).
dbConsistent	Allows consistent updates (Microsoft Jet workspaces only).
dbSQLPassThrough	An SQL pass-through. Causes the SQL statement to be passed to an ODBC database for processing (Microsoft Jet workspaces only).
dbFailOnError	Rolls back updates if an error occurs (Microsoft Jet workspaces only).
dbSeeChanges	Generates a run-time error if another user is changing data you are editing (Microsoft Jet workspaces only).
dbRunAsync	Executes the query asynchronously (ODBCDirect workspaces only).
dbExecDirect	Executes the query without first calling the SQLPrepare ODBC function (ODBCDirect workspaces only).

Idle Method Optional Argument Constant (This Is)

Constant	Description
dbRefreshCache	Forces any pending writes to disk, and refreshes memory from current disk files.

MakeReplica Method Optional Argument Constants (All are)

Constant	Description
dbRepMakePartial	Creates a partial replica.
dbRepMakeReadOnly	Makes replicable elements of new database read-only.

OpenConnection and OpenDatabase Methods Option Argument Constants (All Are)

Constant	Description
dbDriverNoPrompt	The driver manager uses the connection string provided in *connect*. If sufficient information is not provided, a trappable error is returned.
dbDriverPrompt	The driver manager displays the ODBC **Data Sources** dialog box. The connection string used to establish the connection is constructed from the data source name (DSN) selected and completed by the user via the dialog boxes.

dbDriverComplete	If the connection string provided includes the DSN keyword, the driver manager uses the string as provided in connect, otherwise it behaves as it does when **dbDriverPrompt** is specified.
dbDriverCompleteRequired	(Default) Behaves like **dbDriverComplete** except the driver disables the controls for any information not required to complete the connection.

OpenRecordset Method Type Argument Constants (All Are)

Constant	Description
dbOpenDynamic	Opens a dynamic-type **Recordset** (ODBCDirect workspaces only)
dbOpenDynaset	Opens a dynaset-type **Recordset**
dbOpenForwardOnly	Opens a forward-only type **Recordset**
dbOpenSnapshot	Opens a snapshot-type **Recordset**
dbOpenTable	Opens a table-type **Recordset** (Microsoft Jet workspaces only)

OpenRecordset Method LockEdits Argument Constants (All Are)

Constant	Description
dbPessimistic	Pessimistic concurrency. Cursor uses the lowest level of locking sufficient to ensure the record can be updated.
dbReadOnly	Cursor is read-only. No updates are allowed.
dbOptimistic	Optimistic concurrency based on record ID. Cursor compares record ID in old and new records to determine if changes have been made since the record was last accessed.
dbOptimisticValue	Optimistic concurrency based on record values. Cursor compares data values in old and new records to determine if changes have been made since the record was last accessed (ODBCDirect workspaces only).
dbOptimisticBatch	Enables batch optimistic updates (ODBCDirect workspaces only).

OpenRecordset Method Options Argument Constants (All Are)

Constant	Description
dbDenyWrite	Prevents other users from changing

	Recordset records (Microsoft Jet workspaces only).
dbDenyRead	Prevents other users from reading **Recordset** records (table-type in Microsoft Jet workspaces only).
dbReadOnly	Opens the **Recordset** as read-only (Microsoft Jet workspaces only).
dbAppendOnly	Allows user to add new records to the dynaset, but prevents user from reading existing records (dynaset-type in Microsoft Jet workspaces only).
dbInconsistent	Applies updates to all dynaset fields, even if other records are affected (dynaset- and snapshot-type in Microsoft Jet workspaces only).
dbConsistent	Applies updates only to those fields that will not affect other records in the dynaset (dynaset- and snapshot-type in Microsoft Jet workspaces only).
dbSQLPassThrough	Sends an SQL statement to an ODBC database (snapshot-type in Microsoft Jet workspaces only).
dbForwardOnly	Creates a forward-only scrolling snapshot-type **Recordset** (snapshot-type in Microsoft Jet workspaces only).
dbSeeChanges	Generates a run-time error if another user is changing data you are editing (dynaset-type in Microsoft Jet workspaces only).
dbRunAsync	Executes the query asynchronously (ODBCDirect workspaces only).
dbExecDirect	Executes the query without first calling the SQLPrepare ODBC function (ODBCDirect workspaces only).

SetOption Method Parameter Constants (All Are)

Constant	Description
dbPageTimeout	The PageTimeout key
dbSharedAsyncDelay	The SharedAsyncDelay key
dbExclusiveAsyncDelay	The ExclusiveAsyncDelay key
dbLockRetry	The LockRetry key
dbUserCommitSync	The UserCommitSync key
dbImplicitCommitSync	The ImplicitCommitSync key
dbMaxBufferSize	The MaxBufferSize key
dbMaxLocksPerFile	The MaxLocksPerFile key
dbLockDelay	The LockDelay key

dbRecycleLVs	The RecycleLVs key
dbFlushTransactionTimeout	The FlushTransactionTimeout key

Synchronize Method Exchange Argument Constants (All Are)

Constant	Description
dbRepExportChanges	Sends changes from current database to target database.
dbRepImportChanges	Receives changes from target database.
dbRepImpExpChanges	Sends and receives data in a bidirectional exchange.
dbRepSyncInternet	Exchanges data between files connected via an Internet pathway.

Update Method Type Argument Constants (All Are)

Constant	Description
dbUpdateRegular	(Default) Pending changes aren't cached and are written to disk immediately.
dbUpdateBatch	All pending changes in the update cache are written to disk.
dbUpdateCurrentRecord	Only the current record's pending changes are written to disk.

CancelUpdate Method Type Argument Constants (All Are)

Constant	Description
dbUpdateRegular	(Default) Pending changes aren't cached and are written to disk immediately.
dbUpdateBatch	All pending changes in the update cache are written to disk.

Container Object and Containers Collection Example

This example enumerates the **Containers** collection of the Northwind database and the **Properties** collection of each **Container** object in the collection.

```
Sub ContainerObjectX()

   Dim dbsNorthwind As Database
   Dim ctrLoop As Container
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind

      ' Enumerate Containers collection.
      For Each ctrLoop In .Containers
         Debug.Print "Properties of " & ctrLoop.Name _
            & " container"

         ' Enumerate Properties collection of each
         ' Container object.
         For Each prpLoop In ctrLoop.Properties
            Debug.Print "     " & prpLoop.Name _
               & " = " prpLoop
         Next prpLoop

      Next ctrLoop

      .Close
   End With

End Sub
```

DBEngine Object Example

This example enumerates the collections of the **DBEngine** object. See the methods and properties of **DBEngine** for additional examples.

```
Sub DBEngineX()

   Dim wrkLoop As Workspace
   Dim prpLoop As Property

   With DBEngine
      Debug.Print "DBEngine Properties"

      ' Enumerate Properties collection of DBEngine,
      ' trapping for properties whose values are
      ' invalid in this context.
      For Each prpLoop In .Properties
         On Error Resume Next
         Debug.Print "    " & prpLoop.Name & " = " _
            & prpLoop
         On Error GoTo 0
      Next prpLoop

      Debug.Print "Workspaces collection of DBEngine"

      ' Enumerate Workspaces collection of DBEngine.
      For Each wrkLoop In .Workspaces
         Debug.Print "    " & wrkLoop.Name

         ' Enumerate Properties collection of each
         ' Workspace object, trapping for properties
         ' whose values are invalid in this context.
         For Each prpLoop In wrkLoop.Properties
            On Error Resume Next
            Debug.Print "        " & prpLoop.Name & _
               " = " & prpLoop
            On Error GoTo 0
         Next prpLoop

      Next wrkLoop

   End With

End Sub
```

Dynaset-Type Recordset Example

This example opens a dynaset-type **Recordset** and shows the extent to which its fields are updatable.

```
Sub dbOpenDynasetX()

   Dim dbsNorthwind As Database
   Dim rstInvoices As Recordset
   Dim fldLoop As Field

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstInvoices = _
      dbsNorthwind.OpenRecordset("Invoices", dbOpenDynaset)

   With rstInvoices
      Debug.Print "Dynaset-type recordset: " & .Name

      If .Updatable Then
         Debug.Print "    Updatable fields:"

         ' Enumerate Fields collection of dynaset-type
         ' Recordset object, print only updatable
         ' fields.
         For Each fldLoop In .Fields
            If fldLoop.DataUpdatable Then
               Debug.Print "        " & fldLoop.Name
            End If
         Next fldLoop

      End If

      .Close
   End With

   dbsNorthwind.Close

End Sub
```

Field Object, Fields Collection Example

This example shows what properties are valid for a **Field** object depending on where the **Field** resides (for example, the **Fields** collection of a **TableDef**, the **Fields** collection of a **QueryDef**, and so forth). The FieldOutput procedure is required for this procedure to run.

```
Sub FieldX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim fldTableDef As Field
   Dim fldQueryDef As Field
   Dim fldRecordset As Field
   Dim fldRelation As Field
   Dim fldIndex As Field
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")

   ' Assign a Field object from different Fields
   ' collections to object variables.
   Set fldTableDef = _
      dbsNorthwind.TableDefs(0).Fields(0)
   Set fldQueryDef =dbsNorthwind.QueryDefs(0).Fields(0)
   Set fldRecordset = rstEmployees.Fields(0)
   Set fldRelation =dbsNorthwind.Relations(0).Fields(0)
   Set fldIndex = _
      dbsNorthwind.TableDefs(0).Indexes(0).Fields(0)

   ' Print report.
   FieldOutput "TableDef", fldTableDef
   FieldOutput "QueryDef", fldQueryDef
   FieldOutput "Recordset", fldRecordset
   FieldOutput "Relation", fldRelation
   FieldOutput "Index", fldIndex

   rstEmployees.Close
   dbsNorthwind.Close

End Sub

Sub FieldOutput(strTemp As String, fldTemp As Field)
   ' Report function for FieldX.

   Dim prpLoop As Property

   Debug.Print "Valid Field properties in " & strTemp

   ' Enumerate Properties collection of passed Field
   ' object.
   For Each prpLoop In fldTemp.Properties
      ' Some properties are invalid in certain
      ' contexts (the Value property in the Fields
      ' collection of a TableDef for example). Any
      ' attempt to use an invalid property will
```

```
      ' trigger an error.
      On Error Resume Next
      Debug.Print "    " & prpLoop.Name & " = " & _
         prpLoop.Value
      On Error GoTo 0
   Next prpLoop

End Sub
```

Group Object, Groups Collection, User Object, and Users Collection Example

This example illustrates the use of the **Group** and **User** objects and the **Groups** and **Users** collections. First, it creates a new **User** object and appends the object to the **Users** collection of the default **Workspace** object. Next, it creates a new **Group** object and appends the object to the **Groups** collection of the default **Workspace** object. Then the example adds user Pat Smith to the Accounting group. Finally, it enumerates the **Users** and **Groups** collections of the default **Workspace** object. See the methods and properties listed in the **Group** and **User** summary topics for additional examples.

```
Sub GroupX()

   Dim wrkDefault As Workspace
   Dim usrNew As User
   Dim usrLoop As User
   Dim grpNew As Group
   Dim grpLoop As Group
   Dim grpMember As Group

   Set wrkDefault = DBEngine.Workspaces(0)

   With wrkDefault

      ' Create and append new user.
      Set usrNew = .CreateUser("Pat Smith", _
         "abc123DEF456", "Password1")
      .Users.Append usrNew

      ' Create and append new group.
      Set grpNew = .CreateGroup("Accounting", _
         "UVW987xyz654")
      .Groups.Append grpNew

      ' Make the user Pat Smith a member of the
      ' Accounting group by creating and adding the
      ' appropriate Group object to the user's Groups
      ' collection. The same is accomplished if a User
      ' object representing Pat Smith is created and
      ' appended to the Accounting group's Users
      ' collection.
      Set grpMember = usrNew.CreateGroup("Accounting")
      usrNew.Groups.Append grpMember

      Debug.Print "Users collection:"

      ' Enumerate all User objects in the default
      ' workspace's Users collection.
      For Each usrLoop In .Users
         Debug.Print "    " & usrLoop.Name
         Debug.Print "        Belongs to these groups:"

         ' Enumerate all Group objects in each User
         ' object's Groups collection.
         If usrLoop.Groups.Count <> 0 Then
            For Each grpLoop In usrLoop.Groups
               Debug.Print "            " & _
```

```
                  grpLoop.Name
            Next grpLoop
         Else
            Debug.Print "            [None]"
         End If

      Next usrLoop

      Debug.Print "Groups collection:"

      ' Enumerate all Group objects in the default
      ' workspace's Groups collection.
      For Each grpLoop In .Groups
         Debug.Print "    " & grpLoop.Name
         Debug.Print "        Has as its members:"

         ' Enumerate all User objects in each Group
         ' object's Users collection.
         If grpLoop.Users.Count <> 0 Then
            For Each usrLoop In grpLoop.Users
               Debug.Print "            " & _
                  usrLoop.Name
            Next usrLoop
         Else
            Debug.Print "            [None]"
         End If

      Next grpLoop

      ' Delete new User and Group objects because this
      ' is only a demonstration.
      .Users.Delete "Pat Smith"
      .Groups.Delete "Accounting"

   End With

End Sub
```

Index Object, Indexes Collection Example

This example creates a new **Index** object, appends it to the **Indexes** collection of the Employees **TableDef**, and then enumerates the **Indexes** collection of the **TableDef**. Finally, it enumerates a **Recordset**, first using the primary **Index**, and then using the new **Index**. The IndexOutput procedure is required for this procedure to run.

```
Sub IndexObjectX()

   Dim dbsNorthwind As Database
   Dim tdfEmployees As TableDef
   Dim idxNew As Index
   Dim idxLoop As Index
   Dim rstEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set tdfEmployees = dbsNorthwind!Employees

   With tdfEmployees
      ' Create new index, create and append Field
      ' objects to its Fields collection.
      Set idxNew = .CreateIndex("NewIndex")

      With idxNew
         .Fields.Append .CreateField("Country")
         .Fields.Append .CreateField("LastName")
         .Fields.Append .CreateField("FirstName")
      End With

      ' Add new Index object to the Indexes collection
      ' of the Employees table collection.
      .Indexes.Append idxNew
      .Indexes.Refresh

      Debug.Print .Indexes.Count & " Indexes in " & _
         .Name & " TableDef"

      ' Enumerate Indexes collection of Employees
      ' table.
      For Each idxLoop In .Indexes
         Debug.Print "    " & idxLoop.Name
      Next idxLoop

      Set rstEmployees = _
         dbsNorthwind.OpenRecordset("Employees")

      ' Print report using old and new indexes.
      IndexOutput rstEmployees, "PrimaryKey"
      IndexOutput rstEmployees, idxNew.Name
      rstEmployees.Close

      ' Delete new Index because this is a
      ' demonstration.
      .Indexes.Delete idxNew.Name
   End With

   dbsNorthwind.Close
```

```
End Sub

Sub IndexOutput(rstTemp As Recordset, _
   strIndex As String)
   ' Report function for FieldX.

   With rstTemp
      ' Set the index.
      .Index = strIndex
      .MoveFirst
      Debug.Print "Recordset = " & .Name & _
         ", Index = " & .Index
      Debug.Print "    EmployeeID - Country - Name"

      ' Enumerate the recordset using the specified
      ' index.
      Do While Not .EOF
         Debug.Print "    " & !EmployeeID & " - " & _
            !Country & " - " & !LastName & ", " & !FirstName
         .MoveNext
      Loop

   End With

End Sub
```

Parameter Object, Parameters Collection Example

This example demonstrates **Parameter** objects and the **Parameters** collection by creating a temporary **QueryDef** and retrieving data based on changes made to the **QueryDef** object's **Parameters**. The ParametersChange procedure is required for this procedure to run.

```
Sub ParameterX()

   Dim dbsNorthwind As Database
   Dim qdfReport As QueryDef
   Dim prmBegin As Parameter
   Dim prmEnd As Parameter

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Create temporary QueryDef object with two
   ' parameters.
   Set qdfReport = dbsNorthwind.CreateQueryDef("", _
      "PARAMETERS dteBegin DateTime, dteEnd DateTime; " & _
      "SELECT EmployeeID, COUNT(OrderID) AS NumOrders " & _
      "FROM Orders WHERE ShippedDate BETWEEN " & _
      "[dteBegin] AND [dteEnd] GROUP BY EmployeeID " & _
      "ORDER BY EmployeeID")
   Set prmBegin = qdfReport.Parameters!dteBegin
   Set prmEnd = qdfReport.Parameters!dteEnd

   ' Print report using specified parameter values.
   ParametersChange qdfReport, prmBegin, #1/1/95#, _
      prmEnd, #6/30/95#
   ParametersChange qdfReport, prmBegin, #7/1/95#, _
      prmEnd, #12/31/95#

   dbsNorthwind.Close

End Sub

Sub ParametersChange(qdfTemp As QueryDef, _
   prmFirst As Parameter, dteFirst As Date, _
   prmLast As Parameter, dteLast As Date)
   ' Report function for ParameterX.

   Dim rstTemp As Recordset
   Dim fldLoop As Field

   ' Set parameter values and open recordset from
   ' temporary QueryDef object.
   prmFirst = dteFirst
   prmLast = dteLast
   Set rstTemp = _
      qdfTemp.OpenRecordset(dbOpenForwardOnly)
   Debug.Print "Period " & dteFirst & " to " & dteLast

   ' Enumerate recordset.
   Do While Not rstTemp.EOF

      ' Enumerate Fields collection of recordset.
      For Each fldLoop In rstTemp.Fields
```

```
            Debug.Print " - " & fldLoop.Name & " = " & fldLoop;
        Next fldLoop

        Debug.Print
        rstTemp.MoveNext
    Loop

    rstTemp.Close

End Sub
```

Property Object, Properties Collection Example

This example creates a user-defined property for the current database, sets its **Type** and **Value** properties, and appends it to the **Properties** collection of the database. Then the example enumerates all properties in the database. See the properties listed in the **Property** summary topic for additional examples.

```
Sub PropertyX()

   Dim dbsNorthwind As Database
   Dim prpNew As Property
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      ' Create and append user-defined property.
      Set prpNew = .CreateProperty()
      prpNew.Name = "UserDefined"
      prpNew.Type = dbText
      prpNew.Value = "This is a user-defined property."
      .Properties.Append prpNew

      ' Enumerate all properties of current database.
      Debug.Print "Properties of " & .Name
      For Each prpLoop In .Properties
         With prpLoop
            Debug.Print "    " & .Name
            Debug.Print "        Type: " & .Type
            Debug.Print "        Value: " & .Value
            Debug.Print "        Inherited: " & _
               .Inherited
         End With
      Next prpLoop

      ' Delete new property because this is a
      ' demonstration.
      .Properties.Delete "UserDefined"
   End With

End Sub
```

QueryDef Object, QueryDefs Collection Example

This example creates a new **QueryDef** object and appends it to the **QueryDefs** collection of the Northwind **Database** object. It then enumerates the **QueryDefs** collection and the **Properties** collection of the new **QueryDef**.

```
Sub QueryDefX()

   Dim dbsNorthwind As Database
   Dim qdfNew As QueryDef
   Dim qdfLoop As QueryDef
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Create new QueryDef object. Because it has a
   ' name, it is automatically appended to the
   ' QueryDefs collection.
   Set qdfNew = dbsNorthwind.CreateQueryDef("NewQueryDef", _
         "SELECT * FROM Categories")

   With dbsNorthwind
      Debug.Print .QueryDefs.Count & _
         " QueryDefs in " & .Name

      ' Enumerate QueryDefs collection.
      For Each qdfLoop In .QueryDefs
         Debug.Print "     " & qdfLoop.Name
      Next qdfLoop

      With qdfNew
         Debug.Print "Properties of " & .Name

         ' Enumerate Properties collection of new
         ' QueryDef object.
         For Each prpLoop In .Properties
            On Error Resume Next
            Debug.Print "     " & prpLoop.Name & " - " & _
               IIf(prpLoop = "", "[empty]", prpLoop)
            On Error Goto 0
         Next prpLoop
      End With

      ' Delete new QueryDef because this is a
      ' demonstration.
      .QueryDefs.Delete qdfNew.Name
      .Close
   End With

End Sub
```

Recordset Object, Recordsets Collection Example

This example demonstrates **Recordset** objects and the **Recordsets** collection by opening four different types of **Recordsets**, enumerating the **Recordsets** collection of the current **Database**, and enumerating the **Properties** collection of each **Recordset**.

```
Sub RecordsetX()

   Dim dbsNorthwind As Database
   Dim rstTable As Recordset
   Dim rstDynaset As Recordset
   Dim rstSnapshot As Recordset
   Dim rstForwardOnly As Recordset
   Dim rstLoop As Recordset
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind

      ' Open one of each type of Recordset object.
      Set rstTable = .OpenRecordset("Categories", _
         dbOpenTable)
      Set rstDynaset = .OpenRecordset("Employees", _
         dbOpenDynaset)
      Set rstSnapshot = .OpenRecordset("Shippers", _
         dbOpenSnapshot)
      Set rstForwardOnly = .OpenRecordset _
         ("Employees", dbOpenForwardOnly)

      Debug.Print "Recordsets in Recordsets " & _
         "collection of dbsNorthwind"

      ' Enumerate Recordsets collection.
      For Each rstLoop In .Recordsets

         With rstLoop
            Debug.Print "    " & .Name

            ' Enumerate Properties collection of each
            ' Recordset object. Trap for any
            ' properties whose values are invalid in
            ' this context.
            For Each prpLoop In .Properties
               On Error Resume Next
               If prpLoop <> "" Then Debug.Print _
                  "        " & prpLoop.Name & _
                  " = " & prpLoop
               On Error GoTo 0
            Next prpLoop

         End With

      Next rstLoop

      rstTable.Close
      rstDynaset.Close
```

```
        rstSnapshot.Close
        rstForwardOnly.Close

        .Close
    End With

End Sub
```

Relation Object, Relations Collection Example

This example shows how an existing **Relation** object can control data entry. The procedure attempts to add a record with a deliberately incorrect CategoryID; this triggers the error-handling routine.

```
Sub RelationX()

   Dim dbsNorthwind As Database
   Dim rstProducts As Recordset
   Dim prpLoop As Property
   Dim fldLoop As Field
   Dim errLoop As Error

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstProducts = dbsNorthwind.OpenRecordset("Products")

   ' Print a report showing all the different parts of
   ' the relation and where each part is stored.
   With dbsNorthwind.Relations!CategoriesProducts
      Debug.Print "Properties of " & .Name & " Relation"
      Debug.Print "    Table = " & .Table
      Debug.Print "    ForeignTable = " & .ForeignTable
      Debug.Print "Fields of " & .Name & " Relation"
      With .Fields!CategoryID
         Debug.Print "    " & .Name
         Debug.Print "        Name = " & .Name
         Debug.Print "        ForeignName = " & .ForeignName
      End With
   End With

   ' Attempt to add a record that violates the relation.
   With rstProducts
      .AddNew
      !ProductName = "Trygve's Lutefisk"
      !CategoryID = 10
      On Error GoTo Err_Relation
      .Update
      On Error GoTo 0
      .Close
   End With

   dbsNorthwind.Close

   Exit Sub

Err_Relation:

   ' Notify user of any errors that result from
   ' the invalid data.
   If DBEngine.Errors.Count > 0 Then
      For Each errLoop In DBEngine.Errors
         MsgBox "Error number: " & errLoop.Number & _
            vbCr & errLoop.Description
      Next errLoop
   End If
```

```
    Resume Next
End Sub
```

Snapshot-Type Recordset Example

This example opens a snapshot-type **Recordset** and demonstrates its read-only characteristics.

```
Sub dbOpenSnapshotX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees", _
      dbOpenSnapshot)

   With rstEmployees
      Debug.Print "Snapshot-type recordset: " & _
         .Name

      ' Enumerate the Properties collection of the
      ' snapshot-type Recordset object, trapping for
      ' any properties whose values are invalid in
      ' this context.
      For Each prpLoop In .Properties
         On Error Resume Next
         Debug.Print "    " & _
            prpLoop.Name & " = " & prpLoop
         On Error Goto 0
      Next prpLoop

      .Close
   End With

   dbsNorthwind.Close

End Sub
```

Table-Type Recordset Example

This example opens a table-type **Recordset**, sets its **Index** property, and enumerates its records.

```
Sub dbOpenTableX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   ' dbOpenTable is default.
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")

   With rstEmployees
      Debug.Print "Table-type recordset: " & .Name

      ' Use predefined index.
      .Index = "LastName"
      Debug.Print "    Index = " & .Index

      ' Enumerate records.
      Do While Not .EOF
         Debug.Print "        " & !LastName & ", " & _
            !FirstName
         .MoveNext
      Loop

      .Close
   End With

   dbsNorthwind.Close

End Sub
```

TableDef Object, TableDefs Collection Example

This example creates a new **TableDef** object and appends it to the **TableDefs** collection of the Northwind **Database** object. It then enumerates the **TableDefs** collection and the **Properties** collection of the new **TableDef**.

```
Sub TableDefX()

   Dim dbsNorthwind As Database
   Dim tdfNew As TableDef
   Dim tdfLoop As TableDef
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Create new TableDef object, append Field objects
   ' to its Fields collection, and append TableDef
   ' object to the TableDefs collection of the
   ' Database object.
   Set tdfNew = dbsNorthwind.CreateTableDef("NewTableDef")
   tdfNew.Fields.Append tdfNew.CreateField("Date", dbDate)
   dbsNorthwind.TableDefs.Append tdfNew

   With dbsNorthwind
      Debug.Print .TableDefs.Count & _
         " TableDefs in " & .Name

      ' Enumerate TableDefs collection.
      For Each tdfLoop In .TableDefs
         Debug.Print "    " & tdfLoop.Name
      Next tdfLoop

      With tdfNew
         Debug.Print "Properties of " & .Name

         ' Enumerate Properties collection of new
         ' TableDef object, only printing properties
         ' with non-empty values.
         For Each prpLoop In .Properties
            Debug.Print "    " & prpLoop.Name & " - " & _
               IIf(prpLoop = "", "[empty]", prpLoop)
         Next prpLoop

      End With

      ' Delete new TableDef since this is a
      ' demonstration.
      .TableDefs.Delete tdfNew.Name
      .Close
   End With

End Sub
```

Connection Object, Connections Collection Example

This example demonstrates the **Connection** object and **Connections** collection by opening a Microsoft Jet **Database** object and two ODBCDirect **Connection** objects and listing the properties available to each object.

```
Sub ConnectionObjectX()

   Dim wrkJet as Workspace
   Dim dbsNorthwind As Database
   Dim wrkODBC As Workspace
   Dim conPubs As Connection
   Dim conPubs2 As Connection
   Dim conLoop As Connection
   Dim prpLoop As Property

   ' Open Microsoft Jet Database object.
   Set wrkJet = CreateWorkspace("NewJetWorkspace", _
      "admin", "", dbUseJet)
   Set dbsNorthwind = wrkJet.OpenDatabase("Northwind.mdb")

   ' Create ODBCDirect Workspace object and open Connection
   ' objects.
   Set wrkODBC = CreateWorkspace("NewODBCWorkspace", _
      "admin", "", dbUseODBC)
   Set conPubs = wrkODBC.OpenConnection("Connection1", , , _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")
   Set conPubs2 = wrkODBC.OpenConnection("Connection2", , _
      True, "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")

   Debug.Print "Database properties:"

   With dbsNorthwind
      ' Enumerate Properties collection of Database object.
      For Each prpLoop In .Properties
         On Error Resume Next
         Debug.Print "    " & prpLoop.Name & " = " & _
            prpLoop.Value
         On Error GoTo 0
      Next prpLoop
   End With

   ' Enumerate the Connections collection.
   For Each conLoop In wrkODBC.Connections
      Debug.Print "Connection properties for " & _
         conLoop.Name & ":"

      With conLoop
         ' Print property values by explicitly calling each
         ' Property object; the Connection object does not
         ' support a Properties collection.
         Debug.Print "    Connect = " & .Connect
         ' Property actually returns a Database object.
         Debug.Print "    Database[.Name] = " & _
            .Database.Name
         Debug.Print "    Name = " & .Name
         Debug.Print "    QueryTimeout = " & .QueryTimeout
```

```
      Debug.Print "    RecordsAffected = " & _
         .RecordsAffected
      Debug.Print "    StillExecuting = " & _
         .StillExecuting
      Debug.Print "    Transactions = " & .Transactions
      Debug.Print "    Updatable = " & .Updatable
   End With

Next conLoop

dbsNorthwind.Close
conPubs.Close
conPubs2.Close
wrkJet.Close
wrkODBC.Close

End Sub
```

Database Object, Databases Collection Example

This example creates a new **Database** object and opens an existing **Database** object in the default **Workspace** object. Then it enumerates the **Database** collection and the **Properties** collection of each **Database** object. See the methods and properties listed in the **Database** summary topic for additional examples.

```
Sub DatabaseObjectX()

   Dim wrkJet As Workspace
   Dim dbsNorthwind As Database
   Dim dbsNew As Database
   Dim dbsLoop As Database
   Dim prpLoop As Property

   Set wrkJet = CreateWorkspace("JetWorkspace", "admin", _
      "", dbUseJet)

   ' Make sure there isn't already a file with the name of
   ' the new database.
   If Dir("NewDB.mdb") <> "" Then Kill "NewDB.mdb"

   ' Create a new database with the specified
   ' collating order.
   Set dbsNew = wrkJet.CreateDatabase("NewDB.mdb", _
      dbLangGeneral)
   Set dbsNorthwind = wrkJet.OpenDatabase("Northwind.mdb")

   ' Enumerate the Databases collection.
   For Each dbsLoop In wrkJet.Databases
      With dbsLoop
         Debug.Print "Properties of " & .Name
         ' Enumerate the Properties collection of each
         ' Database object.
         For Each prpLoop In .Properties
            If prpLoop <> "" Then Debug.Print "    " & _
               prpLoop.Name & " = " & prpLoop
         Next prpLoop
      End With
   Next dbsLoop

   dbsNew.Close
   dbsNorthwind.Close
   wrkJet.Close

End Sub
```

Document Object and Documents Collection Example

This example enumerates the **Documents** collection of the Tables container, and then enumerates the **Properties** collection of the first **Document** object in the collection.

```
Sub DocumentX()

   Dim dbsNorthwind As Database
   Dim docLoop As Document
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind.Containers!Tables
      Debug.Print "Documents in " & .Name & " container"
      ' Enumerate the Documents collection of the Tables
      ' container.
      For Each docLoop In .Documents
         Debug.Print "    " & docLoop.Name
      Next docLoop
      With .Documents(0)
         ' Enumerate the Properties collection of the first.
         ' Document object of the Tables container.
         Debug.Print "Properties of " & .Name & " document"
         On Error Resume Next
         For Each prpLoop In .Properties
            Debug.Print "    " & prpLoop.Name & " = " & _
               prpLoop
         Next prpLoop
         On Error GoTo 0
      End With
   End With

   dbsNorthwind.Close

End Sub
```

Dynamic-Type Recordset Example

This example opens a dynamic-type **Recordset** object and enumerates its records.

```
Sub dbOpenDynamicX()

   Dim wrkMain As Workspace
   Dim conMain As Connection
   Dim qdfTemp As QueryDef
   Dim rstTemp As Recordset
   Dim strSQL As String
   Dim intLoop As Integer

   ' Create ODBC workspace and open connection to
   ' SQL Server database.
   Set wrkMain = CreateWorkspace("ODBCWorkspace", _
      "admin", "", dbUseODBC)
   Set conMain = wrkMain.OpenConnection("Publishers", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")
   ' Open dynamic-type recordset.
   Set rstTemp = _
      conMain.OpenRecordset("authors", _
      dbOpenDynamic)

   With rstTemp
      Debug.Print "Dynamic-type recordset: " & .Name

      ' Enumerate records.
      Do While Not .EOF
         Debug.Print "        " & !au_lname & ", " & _
            !au_fname
         .MoveNext
      Loop

      .Close
   End With

   conMain.Close
   wrkMain.Close

End Sub
```

Forward-Only–Type Recordset Example

This example opens a forward-only-type **Recordset**, demonstrates its read-only characteristics, and steps through the **Recordset** with the **MoveNext** method.

```
Sub dbOpenForwardOnlyX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim fldLoop As Field

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   ' Open a forward-only-type Recordset object. Only the
   ' MoveNext and Move methods may be used to navigate
   ' through the recordset.
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees", _
      dbOpenForwardOnly)

   With rstEmployees
      Debug.Print "Forward-only-type recordset: " & _
         .Name & ", Updatable = " & .Updatable

      Debug.Print "    Field - DataUpdatable"
      ' Enumerate Fields collection, printing the Name and
      ' DataUpdatable properties of each Field object.
      For Each fldLoop In .Fields
         Debug.Print "        " & _
            fldLoop.Name & " - " & fldLoop.DataUpdatable
      Next fldLoop

      Debug.Print "    Data"
      ' Enumerate the recordset.
      Do While Not .EOF
         Debug.Print "        " & !FirstName & " " & _
            !LastName
         .MoveNext
      Loop

      .Close
   End With

   dbsNorthwind.Close

End Sub
```

Workspace Object, Workspaces Collection Example

This example creates a new Microsoft Jet **Workspace** object and a new ODBCDirect **Workspace** object and appends them to the **Workspaces** collection. It then enumerates the **Workspaces** collections and the **Properties** collection of each **Workspace** object. See the methods and properties of the **Workspace** object or **Workspaces** collection for additional examples.

```
Sub WorkspaceX()

   Dim wrkNewJet As Workspace
   Dim wrkNewODBC As Workspace
   Dim wrkLoop As Workspace
   Dim prpLoop As Property

   ' Create a new Microsoft Jet workspace.
   Set wrkNewJet = CreateWorkspace("NewJetWorkspace", _
      "admin", "", dbUseJet)
   Workspaces.Append wrkNewJet

   ' Create a new ODBCDirect workspace.
   Set wrkNewODBC = CreateWorkspace("NewODBCWorkspace", _
      "admin", "", dbUseODBC)
   Workspaces.Append wrkNewODBC

   ' Enumerate the Workspaces collection.
   For Each wrkLoop In Workspaces
      With wrkLoop
         Debug.Print "Properties of " & .Name
         ' Enumerate the Properties collection of the new
         ' Workspace object.
         For Each prpLoop In .Properties
            On Error Resume Next
            If prpLoop <> "" Then Debug.Print "    " & _
               prpLoop.Name & " = " & prpLoop
            On Error GoTo 0
         Next prpLoop
      End With
   Next wrkLoop

   wrkNewJet.Close
   wrkNewODBC.Close

End Sub
```

AddNew Method Example

This example uses the **AddNew** method to create a new record with the specified name. The AddName function is required for this procedure to run.

```
Sub AddNewX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim strFirstName As String
   Dim strLastName As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees", dbOpenDynaset)

   ' 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

      ' Call the function that adds the record.
      AddName rstEmployees, strFirstName, strLastName

      ' Show the newly added data.
      With rstEmployees
         Debug.Print "New record: " & !FirstName & _
            " " & !LastName
         ' Delete new record because this is a demonstration.
         .Delete
      End With

   Else
      Debug.Print _
         "You must input a string for first and last name!"
   End If

   rstEmployees.Close
   dbsNorthwind.Close

End Sub

Function AddName(rstTemp As Recordset, _
   strFirst As String, strLast As String)

   ' Adds a new record to a Recordset using the data passed
   ' by the calling procedure. The new record is then made
   ' the current record.
   With rstTemp
      .AddNew
      !FirstName = strFirst
      !LastName = strLast
```

```
        .Update
        .Bookmark = .LastModified
    End With

End Function
```

Append and Delete Methods Example

This example uses either the **Append** method or the **Delete** method to modify the **Fields** collection of a **TableDef**. The AppendDeleteField procedure is required for this procedure to run.

```
Sub AppendX()

   Dim dbsNorthwind As Database
   Dim tdfEmployees As TableDef
   Dim fldLoop As Field

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set tdfEmployees = dbsNorthwind.TableDefs!Employees

   ' Add three new fields.
   AppendDeleteField tdfEmployees, "APPEND", _
      "E-mail", dbText, 50
   AppendDeleteField tdfEmployees, "APPEND", _
      "Http", dbText, 80
   AppendDeleteField tdfEmployees, "APPEND", _
      "Quota", dbInteger, 5

   Debug.Print "Fields after Append"
   Debug.Print , "Type", "Size", "Name"

   ' Enumerate the Fields collection to show the new fields.
   For Each fldLoop In tdfEmployees.Fields
      Debug.Print , fldLoop.Type, fldLoop.Size, fldLoop.Name
   Next fldLoop

   ' Delete the newly added fields.
   AppendDeleteField tdfEmployees, "DELETE", "E-mail"
   AppendDeleteField tdfEmployees, "DELETE", "Http"
   AppendDeleteField tdfEmployees, "DELETE", "Quota"

   Debug.Print "Fields after Delete"
   Debug.Print , "Type", "Size", "Name"

   ' Enumerate the Fields collection to show that the new
   ' fields have been deleted.
   For Each fldLoop In tdfEmployees.Fields
      Debug.Print , fldLoop.Type, fldLoop.Size, fldLoop.Name
   Next fldLoop

   dbsNorthwind.Close

End Sub

Sub AppendDeleteField(tdfTemp As TableDef, _
   strCommand As String, strName As String, _
   Optional varType, Optional varSize)

   With tdfTemp

      ' Check first to see if the TableDef object is
      ' updatable. If it isn't, control is passed back to
      ' the calling procedure.
```

```
      If .Updatable = False Then
         MsgBox "TableDef not Updatable! " & _
            "Unable to complete task."
         Exit Sub
      End If

      ' Depending on the passed data, append or delete a
      ' field to the Fields collection of the specified
      ' TableDef object.
      If strCommand = "APPEND" Then
         .Fields.Append .CreateField(strName, _
            varType, varSize)
      Else
         If strCommand = "DELETE" Then .Fields.Delete _
            strName
      End If

   End With

End Sub
```

AppendChunk and GetChunk Methods Example

This example uses the **AppendChunk** and **GetChunk** methods to fill an OLE object field with data from another record, 32K at a time. In a real application, one might use a procedure like this to copy an employee record (including the employee's photo) from one table to another. In this example, the record is simply being copied back to same table. Note that all the chunk manipulation takes place within a single **AddNew-Update** sequence.

```
Sub AppendChunkX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim rstEmployees2 As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Open two recordsets from the Employees table.
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees", _
      dbOpenDynaset)
   Set rstEmployees2 = rstEmployees.Clone

   ' Add a new record to the first Recordset and copy the
   ' data from a record in the second Recordset.
   With rstEmployees
      .AddNew
      !FirstName = rstEmployees2!FirstName
      !LastName = rstEmployees2!LastName
      CopyLargeField rstEmployees2!Photo, !Photo
      .Update

      ' Delete new record because this is a demonstration.
      .Bookmark = .LastModified
      .Delete
      .Close
   End With

   rstEmployees2.Close
   dbsNorthwind.Close

End Sub

Function CopyLargeField(fldSource As Field, _
   fldDestination As Field)

   ' Set size of chunk in bytes.
   Const conChunkSize = 32768

   Dim lngOffset As Long
   Dim lngTotalSize As Long
   Dim strChunk As String

   ' Copy the photo from one Recordset to the other in 32K
   ' chunks until the entire field is copied.
   lngTotalSize = fldSource.FieldSize
   Do While lngOffset < lngTotalSize
```

```
      strChunk = fldSource.GetChunk(lngOffset, conChunkSize)
      fldDestination.AppendChunk strChunk
      lngOffset = lngOffset + conChunkSize
   Loop

End Function
```

BeginTrans, CommitTrans, Rollback Methods Example

This example changes the job title of all sales representatives in the Employees table of the database. After the **BeginTrans** method starts a transaction that isolates all the changes made to the Employees table, the **CommitTrans** method saves the changes. Notice that you can use the **Rollback** method to undo changes that you saved using the **Update** method. Furthermore, the main transaction is nested within another transaction that automatically rolls back any changes made by the user during this example.

One or more table pages remain locked while the user decides whether or not to accept the changes. For this reason, this technique isn't recommended but shown only as an example.

```
Sub BeginTransX()

   Dim strName As String
   Dim strMessage As String
   Dim wrkDefault As Workspace
   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset

   ' Get default Workspace.
   Set wrkDefault = DBEngine.Workspaces(0)
   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")

   ' Start of outer transaction.
   wrkDefault.BeginTrans
   ' Start of main transaction.
   wrkDefault.BeginTrans

   With rstEmployees

      ' Loop through recordset and ask user if she wants to
      ' change the title for a specified employee.
      Do Until .EOF
         If !Title = "Sales Representative" Then
            strName = !LastName & ", " & !FirstName
            strMessage = "Employee: " & strName & vbCr & _
               "Change title to Account Executive?"

            ' Change the title for the specified employee.
            If MsgBox(strMessage, vbYesNo) = vbYes Then
               .Edit
               !Title = "Account Executive"
               .Update
            End If
         End If

         .MoveNext
      Loop

      ' Ask if the user wants to commit to all the changes
      ' made above.
      If MsgBox("Save all changes?", vbYesNo) = vbYes Then
         wrkDefault.CommitTrans
      Else
```

```
         wrkDefault.Rollback
      End If

      ' Print current data in recordset.
      .MoveFirst
      Do While Not .EOF
         Debug.Print !LastName & ", " & !FirstName & _
            " - " & !Title
         .MoveNext
      Loop

      ' Roll back any changes made by the user since this is
      ' a demonstration.
      wrkDefault.Rollback
      .Close
   End With

   dbsNorthwind.Close

End Sub
```

Clone 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.

```
Sub CloneX()

   Dim dbsNorthwind As Database
   Dim arstProducts(1 To 3) As Recordset
   Dim intLoop As Integer
   Dim strMessage As String
   Dim strFind As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' If the following SQL statement will be used often,
   ' creating a permanent QueryDef will result in better
   ' performance.
   Set arstProducts(1) = dbsNorthwind.OpenRecordset( _
      "SELECT ProductName FROM Products " & _
      "ORDER BY ProductName", dbOpenSnapshot)

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

   Do While True

      ' Loop through the array so that on each pass, the
      ' user is searching a different copy of the same
      ' Recordset.
      For intLoop = 1 To 3

         ' Ask for search string while showing where the
         ' current record pointer is for each Recordset.
         strMessage = _
            "Recordsets from Products table:" & vbCr & _
            "  1 - Original - Record pointer at " & _
            arstProducts(1)!ProductName & vbCr & _
            "  2 - Clone - Record pointer at " & _
            arstProducts(2)!ProductName & vbCr & _
            "  3 - Clone - Record pointer at " & _
            arstProducts(3)!ProductName & vbCr & _
            "Enter search string for #" & intLoop & ":"
         strFind = Trim(InputBox(strMessage))
         If strFind = "" Then Exit Do

         ' Find the search string; if there's no match, jump
         ' to the last record.
         With arstProducts(intLoop)
            .FindFirst "ProductName >= '" & strFind & "'"
            If .NoMatch Then .MoveLast
         End With

      Next intLoop

   Loop
```

```
    arstProducts(1).Close
    arstProducts(2).Close
    arstProducts(3).Close
    dbsNorthwind.Close

End Sub
```

Close Method Example

This example uses the **Close** method on both **Recordset** and **Database** objects that have been opened. It also demonstrates how closing a **Recordset** will cause unsaved changes to be lost.

```
Sub CloseX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")

   ' Make changes to a record but close the recordset before
   ' saving the changes.
   With rstEmployees
      Debug.Print "Original data"
      Debug.Print "    Name - Extension"
      Debug.Print "    " & !FirstName & " " & _
         !LastName & " - " & !Extension
      .Edit
      !Extension = "9999"
      .Close
   End With

   ' Reopen Recordset to show that the data hasn't
   ' changed.
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")

   With rstEmployees
      Debug.Print "Data after Close"
      Debug.Print "    Name - Extension"
      Debug.Print "    " & !FirstName & " " & _
         !LastName & " - " & !Extension
      .Close
   End With

   dbsNorthwind.Close

End Sub
```

CompactDatabase Method Example

This example uses the **CompactDatabase** method to change the collating order of a database. You cannot use this code in a module belonging to Northwind.mdb.

```
Sub CompactDatabaseX()

   Dim dbsNorthwind As Database

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Show the properties of the original database.
   With dbsNorthwind
      Debug.Print .Name & ", version " & .Version
      Debug.Print "    CollatingOrder = " & .CollatingOrder
      .Close
   End With

   ' Make sure there isn't already a file with the
   ' name of the compacted database.
   If Dir("NwindKorean.mdb") <> "" Then _
      Kill "NwindKorean.mdb"

   ' This statement creates a compact version of the
   ' Northwind database that uses a Korean language
   ' collating order.
   DBEngine.CompactDatabase "Northwind.mdb", _
      "NwindKorean.mdb", dbLangKorean

   Set dbsNorthwind = OpenDatabase("NwindKorean.mdb")

   ' Show the properties of the compacted database.
   With dbsNorthwind
      Debug.Print .Name & ", version " & .Version
      Debug.Print "    CollatingOrder = " & .CollatingOrder
      .Close
   End With

End Sub
```

This example uses the **CompactDatabase** method to change the version of a database. To run this code, you must have a Microsoft Jet version 1.1 database called Nwind11.mdb and you cannot use this code in a module belonging to Nwind11.mdb.

```
Sub CompactDatabaseX2()

   Dim dbsNorthwind As Database
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Nwind11.mdb")

   ' Show the properties of the original database.
   With dbsNorthwind
      Debug.Print .Name & ", version " & .Version
      Debug.Print "    CollatingOrder = " & .CollatingOrder
      .Close
   End With
```

```
   ' Make sure there isn't already a file with the
   ' name of the compacted database.
   If Dir("Nwind20.mdb") <> "" Then _
      Kill "Nwind20.mdb"

   ' This statement creates a compact and encrypted
   ' Microsoft Jet 2.0 version of a Microsoft Jet version
   ' 1.1 database.
   DBEngine.CompactDatabase "Nwind11.mdb", _
      "Nwind20.mdb", , dbEncrypt + dbVersion20

   Set dbsNorthwind = OpenDatabase("Nwind20.mdb")

   ' Show the properties of the compacted database.
   With dbsNorthwind
      Debug.Print .Name & ", version " & .Version
      For Each prpLoop In .Properties
         On Error Resume Next
         If prpLoop <> "" Then Debug.Print "    " & _
            prpLoop.Name & " = " & prpLoop
         On Error GoTo 0
      Next prpLoop
      .Close
   End With

End Sub
```

CreateDatabase Method Example

This example uses **CreateDatabase** to create a new, encrypted **Database** object.

```
Sub CreateDatabaseX()

   Dim wrkDefault As Workspace
   Dim dbsNew As DATABASE
   Dim prpLoop As Property

   ' Get default Workspace.
   Set wrkDefault = DBEngine.Workspaces(0)

   ' Make sure there isn't already a file with the name of
   ' the new database.
   If Dir("NewDB.mdb") <> "" Then Kill "NewDB.mdb"

   ' Create a new encrypted database with the specified
   ' collating order.
   Set dbsNew = wrkDefault.CreateDatabase("NewDB.mdb", _
      dbLangGeneral, dbEncrypt)

   With dbsNew
      Debug.Print "Properties of " & .Name
      ' Enumerate the Properties collection of the new
      ' Database object.
      For Each prpLoop In .Properties
         If prpLoop <> "" Then Debug.Print "    " & _
            prpLoop.Name & " = " & prpLoop
      Next prpLoop
   End With

   dbsNew.Close

End Sub
```

CreateField Method Example

This example uses the **CreateField** method to create three **Fields** for a new **TableDef**. It then displays the properties of those **Field** objects that are automatically set by the **CreateField** method. (Properties whose values are empty at the time of **Field** creation are not shown.)

```
Sub CreateFieldX()

   Dim dbsNorthwind As Database
   Dim tdfNew As TableDef
   Dim fldLoop As Field
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   Set tdfNew = dbsNorthwind.CreateTableDef("NewTableDef")

   ' Create and append new Field objects for the new
   ' TableDef object.
   With tdfNew
      ' The CreateField method will set a default Size
      ' for a new Field object if one is not specified.
      .Fields.Append .CreateField("TextField", dbText)
      .Fields.Append .CreateField("IntegerField", dbInteger)
      .Fields.Append .CreateField("DateField", dbDate)
   End With

   dbsNorthwind.TableDefs.Append tdfNew

   Debug.Print "Properties of new Fields in " & tdfNew.Name

   ' Enumerate Fields collection to show the properties of
   ' the new Field objects.
   For Each fldLoop In tdfNew.Fields
      Debug.Print "    " & fldLoop.Name

      For Each prpLoop In fldLoop.Properties
         ' Properties that are invalid in the context of
         ' TableDefs will trigger an error if an attempt
         ' is made to read their values.
         On Error Resume Next
         Debug.Print "        " & prpLoop.Name & " - " & _
            IIf(prpLoop = "", "[empty]", prpLoop)
         On Error GoTo 0
      Next prpLoop

   Next fldLoop

   ' Delete new TableDef because this is a demonstration.
   dbsNorthwind.TableDefs.Delete tdfNew.Name
   dbsNorthwind.Close

End Sub
```

CreateIndex Method Example

This example uses the **CreateIndex** method to create two new **Index** objects and then appends them to the **Indexes** collection of the Employees **TableDef** object. It then enumerates the **Indexes** collection of the **TableDef** object, the **Fields** collection of the new **Index** objects, and the **Properties** collection of the new **Index** objects. The CreateIndexOutput function is required for this procedure to run.

```
Sub CreateIndexX()

   Dim dbsNorthwind As Database
   Dim tdfEmployees As TableDef
   Dim idxCountry As Index
   Dim idxFirstName As Index
   Dim idxLoop As Index

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set tdfEmployees = dbsNorthwind!Employees

   With tdfEmployees
      ' Create first Index object, create and append Field
      ' objects to the Index object, and then append the
      ' Index object to the Indexes collection of the
      ' TableDef.
      Set idxCountry = .CreateIndex("CountryIndex")
      With idxCountry
         .Fields.Append .CreateField("Country")
         .Fields.Append .CreateField("LastName")
         .Fields.Append .CreateField("FirstName")
      End With
      .Indexes.Append idxCountry

      ' Create second Index object, create and append Field
      ' objects to the Index object, and then append the
      ' Index object to the Indexes collection of the
      ' TableDef.
      Set idxFirstName = .CreateIndex
      With idxFirstName
         .Name = "FirstNameIndex"
         .Fields.Append .CreateField("FirstName")
         .Fields.Append .CreateField("LastName")
      End With
      .Indexes.Append idxFirstName

      ' Refresh collection so that you can access new Index
      ' objects.
      .Indexes.Refresh

      Debug.Print .Indexes.Count & " Indexes in " & _
         .Name & " TableDef"

      ' Enumerate Indexes collection.
      For Each idxLoop In .Indexes
         Debug.Print "    " & idxLoop.Name
      Next idxLoop
```

```
      ' Print report.
      CreateIndexOutput idxCountry
      CreateIndexOutput idxFirstName

      ' Delete new Index objects because this is a
      ' demonstration.
      .Indexes.Delete idxCountry.Name
      .Indexes.Delete idxFirstName.Name
   End With

   dbsNorthwind.Close

End Sub

Function CreateIndexOutput(idxTemp As Index)

   Dim fldLoop As Field
   Dim prpLoop As Property

   With idxTemp
      ' Enumerate Fields collection of Index object.
      Debug.Print "Fields in " & .Name
      For Each fldLoop In .Fields
         Debug.Print "    " & fldLoop.Name
      Next fldLoop

      ' Enumerate Properties collection of Index object.
      Debug.Print "Properties of " & .Name
      For Each prpLoop In .Properties
         Debug.Print "    " & prpLoop.Name & " - " & _
            IIf(prpLoop = "", "[empty]", prpLoop)
      Next prpLoop
   End With

End Function
```

CreateProperty Method Example

This example tries to set the value of a user-defined property. If the property doesn't exist, it uses the **CreateProperty** method to create and set the value of the new property. The SetProperty procedure is required for this procedure to run.

```
Sub CreatePropertyX()

   Dim dbsNorthwind As Database
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Set the Archive property to True.
   SetProperty dbsNorthwind, "Archive", True

   With dbsNorthwind
      Debug.Print "Properties of " & .Name

      ' Enumerate Properties collection of the Northwind
      ' database.
      For Each prpLoop In .Properties
         If prpLoop <> "" Then Debug.Print "    " & _
            prpLoop.Name & " = " & prpLoop
      Next prpLoop

      ' Delete the new property since this is a
      ' demonstration.
      .Properties.Delete "Archive"

      .Close
   End With

End Sub

Sub SetProperty(dbsTemp As Database, strName As String, _
   booTemp As Boolean)

   Dim prpNew As Property
   Dim errLoop As Error

   ' Attempt to set the specified property.
   On Error GoTo Err_Property
   dbsTemp.Properties("strName") = booTemp
   On Error GoTo 0

   Exit Sub

Err_Property:

   ' Error 3270 means that the property was not found.
   If DBEngine.Errors(0).Number = 3270 Then
      ' Create property, set its value, and append it to the
      ' Properties collection.
      Set prpNew = dbsTemp.CreateProperty(strName, _
         dbBoolean, booTemp)
      dbsTemp.Properties.Append prpNew
```

```
      Resume Next
   Else
      ' If different error has occurred, display message.
      For Each errLoop In DBEngine.Errors
         MsgBox "Error number: " & errLoop.Number & vbCr & _
            errLoop.Description
      Next errLoop
      End
   End If

End Sub
```

CreateQueryDef Method Example

This example uses the **CreateQueryDef** method to create and execute both a temporary and a permanent **QueryDef**. The GetrstTemp function is required for this procedure to run.

```
Sub CreateQueryDefX()

   Dim dbsNorthwind As Database
   Dim qdfTemp As QueryDef
   Dim qdfNew As QueryDef

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      ' Create temporary QueryDef.
      Set qdfTemp = .CreateQueryDef("", _
         "SELECT * FROM Employees")
      ' Open Recordset and print report.
      GetrstTemp qdfTemp
      ' Create permanent QueryDef.
      Set qdfNew = .CreateQueryDef("NewQueryDef", _
         "SELECT * FROM Categories")
      ' Open Recordset and print report.
      GetrstTemp qdfNew
      ' Delete new QueryDef because this is a demonstration.
      .QueryDefs.Delete qdfNew.Name
      .Close
   End With

End Sub

Function GetrstTemp(qdfTemp As QueryDef)

   Dim rstTemp As Recordset

   With qdfTemp
      Debug.Print .Name
      Debug.Print "    " & .SQL
      ' Open Recordset from QueryDef.
      Set rstTemp = .OpenRecordset(dbOpenSnapshot)

      With rstTemp
         ' Populate Recordset and print number of records.
         .MoveLast
         Debug.Print "    Number of records = " & _
            .RecordCount
         Debug.Print
         .Close
      End With

   End With

End Function
```

CreateRelation Method Example

This example uses the **CreateRelation** method to create a **Relation** between the Employees **TableDef** and a new **TableDef** called Departments. This example also demonstrates how creating a new **Relation** will also create any necessary **Indexes** in the foreign table (the DepartmentsEmployees **Index** in the Employees table).

```
Sub CreateRelationX()

   Dim dbsNorthwind As Database
   Dim tdfEmployees As TableDef
   Dim tdfNew As TableDef
   Dim idxNew As Index
   Dim relNew As Relation
   Dim idxLoop As Index

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      ' Add new field to Employees table.
      Set tdfEmployees = .TableDefs!Employees
      tdfEmployees.Fields.Append _
         tdfEmployees.CreateField("DeptID", dbInteger, 2)

      ' Create new Departments table.
      Set tdfNew = .CreateTableDef("Departments")

      With tdfNew
         ' Create and append Field objects to Fields
         ' collection of the new TableDef object.
         .Fields.Append .CreateField("DeptID", dbInteger, 2)
         .Fields.Append .CreateField("DeptName", dbText, 20)

         ' Create Index object for Departments table.
         Set idxNew = .CreateIndex("DeptIDIndex")
         ' Create and append Field object to Fields
         ' collection of the new Index object.
         idxNew.Fields.Append idxNew.CreateField("DeptID")
         ' The index in the primary table must be Unique in
         ' order to be part of a Relation.
         idxNew.Unique = True
         .Indexes.Append idxNew
      End With

      .TableDefs.Append tdfNew

      ' Create EmployeesDepartments Relation object, using
      ' the names of the two tables in the relation.
      Set relNew = .CreateRelation("EmployeesDepartments", _
         tdfNew.Name, tdfEmployees.Name, _
         dbRelationUpdateCascade)

      ' Create Field object for the Fields collection of the
      ' new Relation object. Set the Name and ForeignName
      ' properties based on the fields to be used for the
      ' relation.
      relNew.Fields.Append relNew.CreateField("DeptID")
```

```
      relNew.Fields!DeptID.ForeignName = "DeptID"
      .Relations.Append relNew

      ' Print report.
      Debug.Print "Properties of " & relNew.Name & _
         " Relation"
      Debug.Print "    Table = " & relNew.Table
      Debug.Print "    ForeignTable = " & _
         relNew.ForeignTable
      Debug.Print "Fields of " & relNew.Name & " Relation"

      With relNew.Fields!DeptID
         Debug.Print "    " & .Name
         Debug.Print "        Name = " & .Name
         Debug.Print "        ForeignName = " & .ForeignName
      End With

      Debug.Print "Indexes in " & tdfEmployees.Name & _
         " TableDef"
      For Each idxLoop In tdfEmployees.Indexes
         Debug.Print "    " & idxLoop.Name & _
            ", Foreign = " & idxLoop.Foreign
      Next idxLoop

      ' Delete new objects because this is a demonstration.
      .Relations.Delete relNew.Name
      .TableDefs.Delete tdfNew.Name
      tdfEmployees.Fields.Delete "DeptID"
      .Close
   End With

End Sub
```

CreateTableDef Method Example

This example creates a new **TableDef** object in the Northwind database.

```
Sub CreateTableDefX()

   Dim dbsNorthwind As Database
   Dim tdfNew As TableDef
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Create a new TableDef object.
   Set tdfNew = dbsNorthwind.CreateTableDef("Contacts")

   With tdfNew
      ' Create fields and append them to the new TableDef
      ' object. This must be done before appending the
      ' TableDef object to the TableDefs collection of the
      ' Northwind database.
      .Fields.Append .CreateField("FirstName", dbText)
      .Fields.Append .CreateField("LastName", dbText)
      .Fields.Append .CreateField("Phone", dbText)
      .Fields.Append .CreateField("Notes", dbMemo)

      Debug.Print "Properties of new TableDef object " & _
         "before appending to collection:"

      ' Enumerate Properties collection of new TableDef
      ' object.
      For Each prpLoop In .Properties
         On Error Resume Next
         If prpLoop <> "" Then Debug.Print "     " & _
            prpLoop.Name & " = " & prpLoop
         On Error GoTo 0
      Next prpLoop

      ' Append the new TableDef object to the Northwind
      ' database.
      dbsNorthwind.TableDefs.Append tdfNew

      Debug.Print "Properties of new TableDef object " & _
         "after appending to collection:"

      ' Enumerate Properties collection of new TableDef
      ' object.
      For Each prpLoop In .Properties
         On Error Resume Next
         If prpLoop <> "" Then Debug.Print "     " & _
            prpLoop.Name & " = " & prpLoop
         On Error GoTo 0
      Next prpLoop

   End With

   ' Delete new TableDef object since this is a
   ' demonstration.
```

```
    dbsNorthwind.TableDefs.Delete "Contacts"

    dbsNorthwind.Close

End Sub
```

Delete Method Example

This example uses the **Delete** method to remove a specified record from a **Recordset**. The DeleteRecord procedure is required for this procedure to run

```
Sub DeleteX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim lngID As Long

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")

   ' Add temporary record to be deleted.
   With rstEmployees
      .Index = "PrimaryKey"
      .AddNew
      !FirstName = "Janelle"
      !LastName = "Tebbs"
      .Update
      .Bookmark = .LastModified
      lngID = !EmployeeID
   End With

   ' Delete the employee record with the specified ID
   ' number.
   DeleteRecord rstEmployees, lngID

   rstEmployees.Close
   dbsNorthwind.Close

End Sub

Sub DeleteRecord(rstTemp As Recordset, _
   lngSeek As Long)

   With rstTemp
      .Seek "=", lngSeek
      If .NoMatch Then
         MsgBox "No employee #" & lngSeek & " in file!"
      Else
         .Delete
         MsgBox "Record for employee #" & lngSeek & _
            " deleted!"
      End If
   End With

End Sub
```

Edit Method Example

This example uses the **Edit** method to replace the current data with the specified name. The EditName procedure is required for this procedure to run.

```
Sub EditX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim strOldFirst As String
   Dim strOldLast As String
   Dim strFirstName As String
   Dim strLastName As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees", _
      dbOpenDynaset)

   ' Store original data.
   strOldFirst = rstEmployees!FirstName
   strOldLast = rstEmployees!LastName

   ' Get new data for record.
   strFirstName = Trim(InputBox( _
         "Enter first name:"))
   strLastName = Trim(InputBox( _
         "Enter last name:"))

   ' Proceed if the user entered something for both fields.
   If strFirstName <> "" and strLastName <> "" Then
      ' Update record with new data.
      EditName rstEmployees, strFirstName, strLastName

      With rstEmployees
         ' Show old and new data.
         Debug.Print "Old data: " & strOldFirst & _
            " " & strOldLast
         Debug.Print "New data: " & !FirstName & _
            " " & !LastName
         ' Restore original data because this is a
         ' demonstration.
         .Edit
         !FirstName = strOldFirst
         !LastName = strOldLast
         .Update
      End With

   Else
      Debug.Print _
         "You must input a string for first and last name!"
   End If

   rstEmployees.Close
   dbsNorthwind.Close

End Sub
```

```
Sub EditName(rstTemp As Recordset, _
   strFirst As String, strLast As String)

   ' Make changes to record and set the bookmark to keep
   ' the same record current.
   With rstTemp
      .Edit
      !FirstName = strFirst
      !LastName = strLast
      .Update
      .Bookmark = .LastModified
   End With

End Sub
```

Execute Method Example

This example demonstrates the **Execute** method when run from both a **QueryDef** object and a **Database** object. The ExecuteQueryDef and PrintOutput procedures are required for this procedure to run.

```
Sub ExecuteX()

   Dim dbsNorthwind As Database
   Dim strSQLChange As String
   Dim strSQLRestore As String
   Dim qdfChange As QueryDef
   Dim rstEmployees As Recordset
   Dim errLoop As Error

   ' Define two SQL statements for action queries.
   strSQLChange = "UPDATE Employees SET Country = " & _
      "'United States' WHERE Country = 'USA'"
   strSQLRestore = "UPDATE Employees SET Country = " & _
      "'USA' WHERE Country = 'United States'"

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   ' Create temporary QueryDef object.
   Set qdfChange = dbsNorthwind.CreateQueryDef("", _
      strSQLChange)
   Set rstEmployees = dbsNorthwind.OpenRecordset( _
      "SELECT LastName, Country FROM Employees", _
      dbOpenForwardOnly)

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

   ' Run temporary QueryDef.
   ExecuteQueryDef qdfChange, rstEmployees

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

   ' Run action query to restore data. Trap for errors,
   ' checking the Errors collection if necessary.
   On Error GoTo Err_Execute
   dbsNorthwind.Execute strSQLRestore, dbFailOnError
   On Error GoTo 0

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

   ' Print report of restored data.
   Debug.Print "Data after executing the query " & _
      "to restore the original information"
   PrintOutput rstEmployees

   rstEmployees.Close
```

```
   Exit Sub

Err_Execute:

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

   Resume Next

End Sub

Sub ExecuteQueryDef(qdfTemp As QueryDef, _
   rstTemp As Recordset)

   Dim errLoop As Error

   ' Run the specified QueryDef object. Trap for errors,
   ' checking the Errors collection if necessary.
   On Error GoTo Err_Execute
   qdfTemp.Execute dbFailOnError
   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 DBEngine.Errors.Count > 0 Then
      For Each errLoop In DBEngine.Errors
         MsgBox "Error number: " & errLoop.Number & vbCr & _
            errLoop.Description
      Next errLoop
   End If

   Resume Next

End Sub

Sub PrintOutput(rstTemp As Recordset)

   ' Enumerate Recordset.
   Do While Not rstTemp.EOF
      Debug.Print "    " & rstTemp!LastName & _
         ", " & rstTemp!Country
      rstTemp.MoveNext
   Loop
```

```
End Sub
```

FindFirst, FindLast, FindNext, FindPrevious Methods Example

This example uses the **FindFirst**, **FindLast**, **FindNext**, and **FindPrevious** methods to move the record pointer of a **Recordset** based on the supplied search string and command. The FindAny function is required for this procedure to run.

```
Sub FindFirstX()

   Dim dbsNorthwind As Database
   Dim rstCustomers As Recordset
   Dim strCountry As String
   Dim varBookmark As Variant
   Dim strMessage As String
   Dim intCommand As Integer

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstCustomers = dbsNorthwind.OpenRecordset( _
      "SELECT CompanyName, City, Country " & _
      "FROM Customers ORDER BY CompanyName", _
      dbOpenSnapshot)

   Do While True
      ' Get user input and build search string.
      strCountry = _
         Trim(InputBox("Enter country for search."))
      If strCountry = "" Then Exit Do
      strCountry = "Country = '" & strCountry & "'"

      With rstCustomers
         ' Populate recordset.
         .MoveLast
         ' Find first record satisfying search string. Exit
         ' loop if no such record exists.
         .FindFirst strCountry
         If .NoMatch Then
            MsgBox "No records found with " & _
               strCountry & "."
            Exit Do
         End If

         Do While True
            ' Store bookmark of current record.
            varBookmark = .Bookmark
            ' Get user choice of which method to use.
            strMessage = "Company: " & !CompanyName & _
               vbCr & "Location: " & !City & ", " & _
               !Country & vbCr & vbCr & _
               strCountry & vbCr & vbCr & _
               "[1 - FindFirst, 2 - FindLast, " & _
               vbCr & "3 - FindNext, " & _
               "4 - FindPrevious]"
            intCommand = Val(Left(InputBox(strMessage), 1))
            If intCommand < 1 Or intCommand > 4 Then Exit Do

            ' Use selected Find method. If the Find fails,
            ' return to the last current record.
            If FindAny(intCommand, rstCustomers, _
```

```
                  strCountry) = False Then
               .Bookmark = varBookmark
               MsgBox "No match--returning to " & _
                  "current record."
            End If

         Loop

      End With

      Exit Do
   Loop

   rstCustomers.Close
   dbsNorthwind.Close

End Sub

Function FindAny(intChoice As Integer, _
   rstTemp As Recordset, _
   strFind As String) As Boolean

   ' Use Find method based on user input.
   Select Case intChoice
      Case 1
         rstTemp.FindFirst strFind
      Case 2
         rstTemp.FindLast strFind
      Case 3
         rstTemp.FindNext strFind
      Case 4
         rstTemp.FindPrevious strFind
   End Select

   ' Set return value based on NoMatch property.
   FindAny = IIf(rstTemp.NoMatch, False, True)

End Function
```

GetRows 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.

```
Sub GetRowsX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim strMessage As String
   Dim intRows As Integer
   Dim avarRecords As Variant
   Dim intRecord As Integer

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = dbsNorthwind.OpenRecordset( _
      "SELECT FirstName, LastName, Title " & _
      "FROM Employees ORDER BY LastName", dbOpenSnapshot)

   With rstEmployees
      Do While True
         ' Get user input for number of rows.
         strMessage = "Enter number of rows to retrieve."
         intRows = Val(InputBox(strMessage))

         If intRows <= 0 Then Exit Do

         ' If GetRowsOK is successful, print the results,
         ' noting if the end of the file was reached.
         If GetRowsOK(rstEmployees, intRows, _
               avarRecords) Then
            If intRows > UBound(avarRecords, 2) + 1 Then
               Debug.Print "(Not enough records in " & _
                  "Recordset to retrieve " & intRows & _
                  " rows.)"
            End If
            Debug.Print UBound(avarRecords, 2) + 1 & _
               " records found."

            ' Print the retrieved data.
            For intRecord = 0 To UBound(avarRecords, 2)
               Debug.Print "    " & _
                  avarRecords(0, intRecord) & " " & _
                  avarRecords(1, intRecord) & ", " & _
                  avarRecords(2, intRecord)
            Next intRecord
         Else
            ' Assuming the GetRows error was due to data
            ' changes by another user, use Requery to
            ' refresh the Recordset and start over.
            If .Restartable Then
               If MsgBox("GetRows failed--retry?", _
                     vbYesNo) = vbYes Then
```

```
               .Requery
            Else
               Debug.Print "GetRows failed!"
               Exit Do
            End If
         Else
            Debug.Print "GetRows failed! " & _
               "Recordset not Restartable!"
            Exit Do
         End If
      End If

      ' Because using GetRows leaves the current record
      ' pointer at the last record accessed, move the
      ' pointer back to the beginning of the Recordset
      ' before looping back for another search.
      .MoveFirst
   Loop
End With

rstEmployees.Close
dbsNorthwind.Close

End Sub

Function GetRowsOK(rstTemp As Recordset, _
   intNumber As Integer, avarData As Variant) As Boolean

   ' Store results of GetRows method in array.
   avarData = rstTemp.GetRows(intNumber)
   ' Return False only if fewer than the desired number of
   ' rows were returned, but not because the end of the
   ' Recordset was reached.
   If intNumber > UBound(avarData, 2) + 1 And _
         Not rstTemp.EOF Then
      GetRowsOK = False
   Else
      GetRowsOK = True
   End If

End Function
```

Move Method Example

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

```
Sub MoveX()

   Dim dbsNorthwind As Database
   Dim rstSuppliers As Recordset
   Dim varBookmark As Variant
   Dim strCommand As String
   Dim lngMove As Long

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstSuppliers = _
      dbsNorthwind.OpenRecordset("SELECT CompanyName, " & _
      "City, Country FROM Suppliers ORDER BY CompanyName", _
      dbOpenDynaset)

   With rstSuppliers
      ' Populate recordset.
      .MoveLast
      .MoveFirst

      Do While True
         ' Display information about current record and ask
         ' how many records to move.
         strCommand = InputBox( _
            "Record " & (.AbsolutePosition + 1) & " of " & _
            .RecordCount & vbCr & "Company: " & _
            !CompanyName & vbCr & "Location: " & !City & _
            ", " & !Country & vbCr & vbCr & _
            "Enter number of records to Move " & _
            "(positive or negative).")

         If strCommand = "" Then Exit Do

         ' Store bookmark in case the Move doesn't work.
         varBookmark = .Bookmark

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

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

```
    dbsNorthwind.Close

End Sub
```

MoveFirst, MoveLast, MoveNext, 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.

```
Sub MoveFirstX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim strMessage As String
   Dim intCommand As Integer

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = dbsNorthwind.OpenRecordset( _
      "SELECT FirstName, LastName FROM Employees " & _
      "ORDER BY LastName", dbOpenSnapshot)

   With rstEmployees
      ' Populate Recordset.
      .MoveLast
      .MoveFirst
      Do While True
         ' Show current record information and get user's
         ' method choice.
         strMessage = "Name: " & !FirstName & " " & _
            !LastName & vbCr & "Record " & _
            (.AbsolutePosition + 1) & " of " & _
            .RecordCount & vbCr & vbCr & _
            "[1 - MoveFirst, 2 - MoveLast, " & vbCr & _
            "3 - MoveNext, 4 - MovePrevious]"
         intCommand = Val(Left(InputBox(strMessage), 1))
         If intCommand < 1 Or intCommand > 4 Then Exit Do

         ' Call method based on user's input.
         MoveAny intCommand, rstEmployees
      Loop
      .Close
   End With

   dbsNorthwind.Close

End Sub

Sub MoveAny(intChoice As Integer, _
   rstTemp As Recordset)

   ' Use specified method, trapping for BOF and EOF.
   With rstTemp
      Select Case intChoice
         Case 1
            .MoveFirst
         Case 2
            .MoveLast
         Case 3
            .MoveNext
            If .EOF Then
```

```
            MsgBox "Already at end of recordset!"
            .MoveLast
         End If
      Case 4
         .MovePrevious
         If .BOF Then
            MsgBox "Already at beginning of recordset!"
            .MoveFirst
         End If
   End Select
End With

End Sub
```

NewPassword Method Example

This example asks the user for a new password for user Pat Smith. If the input is a string between 1 and 14 characters long, the example uses the **NewPassword** method to change the password. The user must be logged on as Pat Smith or as a member of the Admins group.

```
Sub NewPasswordX()

   Dim wrkDefault As Workspace
   Dim usrNew As User
   Dim grpNew As Group
   Dim grpMember As Group
   Dim strPassword As String

   ' Get default workspace.
   Set wrkDefault = DBEngine.Workspaces(0)

   With wrkDefault

      ' Create and append new user.
      Set usrNew = .CreateUser("Pat Smith", _
         "abc123DEF456", "Password1")
      .Users.Append usrNew

      ' Create and append new group.
      Set grpNew = .CreateGroup("Accounting", _
         "UVW987xyz654")
      .Groups.Append grpNew

      ' Make the new user a member of the new group.
      Set grpMember = usrNew.CreateGroup("Accounting")
      usrNew.Groups.Append grpMember

      ' Ask user for new password. If input is too long, ask
      ' again.
      Do While True
         strPassword = InputBox("Enter new password:")
         Select Case Len(strPassword)
            Case 1 To 14
               usrNew.NewPassword "Password1", strPassword
               MsgBox "Password changed!"
               Exit Do
            Case Is > 14
               MsgBox "Password too long!"
            Case 0
               Exit Do
         End Select
      Loop

      ' Delete new User and Group objects because this
      ' is only a demonstration.
      .Users.Delete "Pat Smith"
      .Groups.Delete "Accounting"

   End With

End Sub
```

Refresh Method Example

This example uses the **Refresh** method to update the **Fields** collection of the Categories table based on changes to the **OrdinalPosition** data. The order of the **Fields** in the collection changes only after the **Refresh** method is used.

```
Sub RefreshX()

   Dim dbsNorthwind As Database
   Dim tdfEmployees As TableDef
   Dim aintPosition() As Integer
   Dim astrFieldName() As String
   Dim intTemp As Integer
   Dim fldLoop As Field

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set tdfEmployees = dbsNorthwind.TableDefs("Categories")

   With tdfEmployees
      ' Display original OrdinalPosition data and store it
      ' in an array.
      Debug.Print _
         "Original OrdinalPosition data in TableDef."
      ReDim aintPosition(0 To .Fields.Count - 1) As Integer
      ReDim astrFieldName(0 To .Fields.Count - 1) As String
      For intTemp = 0 To .Fields.Count - 1
         aintPosition(intTemp) = _
            .Fields(intTemp).OrdinalPosition
         astrFieldName(intTemp) = .Fields(intTemp).Name
         Debug.Print , aintPosition(intTemp), _
            astrFieldName(intTemp)
      Next intTemp

      ' Change OrdinalPosition data.
      For Each fldLoop In .Fields
         fldLoop.OrdinalPosition = _
            100 - fldLoop.OrdinalPosition
      Next fldLoop
      Set fldLoop = Nothing

      ' Print new data.
      Debug.Print "New OrdinalPosition data before Refresh."
      For Each fldLoop In .Fields
         Debug.Print , fldLoop.OrdinalPosition, fldLoop.Name
      Next fldLoop

      .Fields.Refresh

      ' Print new data, showing how the field order has been
      ' changed.
      Debug.Print "New OrdinalPosition data after Refresh."
      For Each fldLoop In .Fields
         Debug.Print , fldLoop.OrdinalPosition, fldLoop.Name
      Next fldLoop

      ' Restore original OrdinalPosition data.
      For intTemp = 0 To .Fields.Count - 1
```

```
            .Fields(astrFieldName(intTemp)).OrdinalPosition = _
               aintPosition(intTemp)
         Next intTemp
      End With

      dbsNorthwind.Close

   End Sub
```

RepairDatabase Method Example

This example attempts to repair the database named Northwind.mdb. You cannot run this procedure from a module within Northwind.mdb.

```
Sub RepairDatabaseX()

   Dim errLoop As Error

   If MsgBox("Repair the Northwind database?", _
         vbYesNo) = vbYes Then
      On Error GoTo Err_Repair
      DBEngine.RepairDatabase "Northwind.mdb"
      On Error GoTo 0
      MsgBox "End of repair procedure!"
   End If

   Exit Sub

Err_Repair:

   For Each errLoop In DBEngine.Errors
      MsgBox "Repair unsuccessful!" & vbCr & _
         "Error number: " & errLoop.Number & _
         vbCr & errLoop.Description
   Next errLoop

End Sub
```

Seek Method Example

This example demonstrates the **Seek** method by allowing the user to search for a product based on an ID number.

```
Sub SeekX()

   Dim dbsNorthwind As Database
   Dim rstProducts As Recordset
   Dim intFirst As Integer
   Dim intLast As Integer
   Dim strMessage As String
   Dim strSeek As String
   Dim varBookmark As Variant

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   ' You must open a table-type Recordset to use an index,
   ' and hence the Seek method.
   Set rstProducts = _
      dbsNorthwind.OpenRecordset("Products", dbOpenTable)

   With rstProducts
      ' Set the index.
      .Index = "PrimaryKey"

      ' Get the lowest and highest product IDs.
      .MoveLast
      intLast = !ProductID
      .MoveFirst
      intFirst = !ProductID

      Do While True
         ' Display current record information and ask user
         ' for ID number.
         strMessage = "Product ID: " & !ProductID & vbCr & _
            "Name: " & !ProductName & vbCr & vbCr & _
            "Enter a product ID between " & intFirst & _
            " and " & intLast & "."
         strSeek = InputBox(strMessage)

         If strSeek = "" Then Exit Do

         ' Store current bookmark in case the Seek fails.
         varBookmark = .Bookmark

         .Seek "=", Val(strSeek)

         ' Return to the current record if the Seek fails.
         If .NoMatch Then
            MsgBox "ID not found!"
            .Bookmark = varBookmark
         End If
      Loop

      .Close
   End With
```

```
    dbsNorthwind.Close

End Sub
```

Update Method Example

This example demonstrates the **Update** method in conjunction with **Edit** method.

```
Sub UpdateX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim strOldFirst As String
   Dim strOldLast As String
   Dim strMessage As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")

   With rstEmployees
      .Edit
      ' Store original data.
      strOldFirst = !FirstName
      strOldLast = !LastName
      ' Change data in edit buffer.
      !FirstName = "Linda"
      !LastName = "Kobara"

      ' Show contents of buffer and get user input.
      strMessage = "Edit in progress:" & vbCr & _
         "    Original data = " & strOldFirst & " " & _
         strOldLast & vbCr & "    Data in buffer = " & _
         !FirstName & " " & !LastName & vbCr & vbCr & _
         "Use Update to replace the original data with " & _
         "the buffered data in the Recordset?"

      If MsgBox(strMessage, vbYesNo) = vbYes Then
         .Update
      Else
         .CancelUpdate
      End If

      ' Show the resulting data.
      MsgBox "Data in recordset = " & !FirstName & " " & _
         !LastName

      ' Restore original data because this is a demonstration.
      If Not (strOldFirst = !FirstName And _
            strOldLast = !LastName) Then
         .Edit
         !FirstName = strOldFirst
         !LastName = strOldLast
         .Update
      End If

      .Close
   End With

   dbsNorthwind.Close
```

```
End Sub
```

This example demonstrates the **Update** method in conjunction with the **AddNew** method.

```
Sub UpdateX2()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim strOldFirst As String
   Dim strOldLast As String
   Dim strMessage As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")

   With rstEmployees
      .AddNew
      !FirstName = "Bill"
      !LastName = "Sornsin"

      ' Show contents of buffer and get user input.
      strMessage = "AddNew in progress:" & vbCr & _
         "    Data in buffer = " & !FirstName & " " & _
         !LastName & vbCr & vbCr & _
         "Use Update to save buffer to recordset?"

      If MsgBox(strMessage, vbYesNoCancel) = vbYes Then
         .Update
         ' Go to the new record and show the resulting data.
         .Bookmark = .LastModified
         MsgBox "Data in recordset = " & !FirstName & _
            " " & !LastName
         ' Delete new data because this is a demonstration.
         .Delete
      Else
         .CancelUpdate
         MsgBox "No new record added."
      End If

      .Close
   End With

   dbsNorthwind.Close

End Sub
```

CancelUpdate Method Example

This example shows how the **CancelUpdate** method is used with the **AddNew** method.

```
Sub CancelUpdateX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim intCommand As Integer

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = dbsNorthwind.OpenRecordset( _
      "Employees", dbOpenDynaset)

   With rstEmployees
      .AddNew
      !FirstName = "Kimberly"
      !LastName = "Bowen"
      intCommand = MsgBox("Add new record for " & _
         !FirstName & " " & !LastName & "?", vbYesNo)
      If intCommand = vbYes Then
         .Update
         MsgBox "Record added."
         ' Delete new record because this is a
         ' demonstration.
         .Bookmark = .LastModified
         .Delete
      Else
         .CancelUpdate
         MsgBox "Record not added."
      End If
   End With

   dbsNorthwind.Close

End Sub
```

This example shows how the **CancelUpdate** method is used with the **Edit** method.

```
Sub CancelUpdateX2()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim strFirst As String
   Dim strLast As String
   Dim intCommand As Integer

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = dbsNorthwind.OpenRecordset( _
      "Employees", dbOpenDynaset)

   With rstEmployees
      strFirst = !FirstName
      strLast = !LastName
      .Edit
      !FirstName = "Cora"
      !LastName = "Edmonds"
      intCommand = MsgBox("Replace current name with " & _
```

```
         !FirstName & " " & !LastName & "?", vbYesNo)
      If intCommand = vbYes Then
         .Update
         MsgBox "Record modified."
         ' Restore data because this is a demonstration.
         .Bookmark = .LastModified
         .Edit
         !FirstName = strFirst
         !LastName = strLast
         .Update
      Else
         .CancelUpdate
         MsgBox "Record not modified."
      End If
      .Close
   End With

   dbsNorthwind.Close

End Sub
```

CopyQueryDef Method Example

This example uses the **CopyQueryDef** method to create a copy of a **QueryDef** from an existing **Recordset** and modifies the copy by adding a clause to the **SQL** property. When you create a permanent **QueryDef,** spaces, semicolons, or linefeeds may be added to the **SQL** property; these extra characters must be stripped before any new clauses can be attached to the SQL statement.

```
Function CopyQueryNew(rstTemp As Recordset, _
   strAdd As String) As QueryDef

   Dim strSQL As String
   Dim strRightSQL As String

   Set CopyQueryNew = rstTemp.CopyQueryDef
   With CopyQueryNew
      ' Strip extra characters.
      strSQL = .SQL
      strRightSQL = Right(strSQL, 1)
      Do While strRightSQL = " " Or strRightSQL = ";" Or _
            strRightSQL = Chr(10) Or strRightSQL = vbCr
         strSQL = Left(strSQL, Len(strSQL) - 1)
         strRightSQL = Right(strSQL, 1)
      Loop
      .SQL = strSQL & strAdd
   End With

End Function
```

This example shows a possible use of CopyQueryNew().

```
Sub CopyQueryDefX()

   Dim dbsNorthwind As Database
   Dim qdfEmployees As QueryDef
   Dim rstEmployees As Recordset
   Dim intCommand As Integer
   Dim strOrderBy As String
   Dim qdfCopy As QueryDef
   Dim rstCopy As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set qdfEmployees = dbsNorthwind.CreateQueryDef( _
      "NewQueryDef", "SELECT FirstName, LastName, " & _
      "BirthDate FROM Employees")
   Set rstEmployees = qdfEmployees.OpenRecordset( _
      dbOpenForwardOnly)

   Do While True
      intCommand = Val(InputBox( _
         "Choose field on which to order a new " & _
         "Recordset:" & vbCr & "1 - FirstName" & vbCr & _
         "2 - LastName" & vbCr & "3 - BirthDate" & vbCr & _
         "[Cancel - exit]"))
      Select Case intCommand
         Case 1
            strOrderBy = " ORDER BY FirstName"
         Case 2
```

```
            strOrderBy = " ORDER BY LastName"
         Case 3
            strOrderBy = " ORDER BY BirthDate"
         Case Else
            Exit Do
      End Select
      Set qdfCopy = CopyQueryNew(rstEmployees, strOrderBy)
      Set rstCopy = qdfCopy.OpenRecordset(dbOpenSnapshot, _
         dbForwardOnly)
      With rstCopy
         Do While Not .EOF
            Debug.Print !LastName & ", " & !FirstName & _
               " - " & !BirthDate
            .MoveNext
         Loop
         .Close
      End With
      Exit Do
   Loop

   rstEmployees.Close
   ' Delete new QueryDef because this is a demonstration.
   dbsNorthwind.QueryDefs.Delete qdfEmployees.Name
   dbsNorthwind.Close

End Sub
```

CreateGroup Method Example

This example uses the **CreateGroup** method to create a new **Group** object; it then makes the "admin" user a member of the new **Group** object and lists its properties and users.

```
Sub CreateGroupX()

   Dim wrkDefault As Workspace
   Dim grpNew As Group
   Dim grpTemp As Group
   Dim prpLoop As Property
   Dim usrLoop As User

   Set wrkDefault = DBEngine.Workspaces(0)

   With wrkDefault

      ' Create and append new group.
      Set grpNew = .CreateGroup("NewGroup", _
         "AAA123456789")
      .Groups.Append grpNew

      ' Make the user "admin" a member of the
      ' group NewGroup by creating and adding the
      ' appropriate Group object to the user's Groups
      ' collection.
      Set grpTemp = .Users("admin").CreateGroup("NewGroup")
      .Users("admin").Groups.Append grpTemp

      Debug.Print "Properties of " & grpNew.Name

      ' Enumerate the Properties collection of NewGroup. The
      ' PID property is not readable.
      For Each prpLoop In grpNew.Properties
         On Error Resume Next
         If prpLoop <> "" Then Debug.Print "    " & _
            prpLoop.Name & " = " & prpLoop
         On Error GoTo 0
      Next prpLoop

      Debug.Print "Users collection of " & grpNew.Name

      ' Enumerate the Users collection of NewGroup.
      For Each usrLoop In grpNew.Users
         Debug.Print "    " & _
            usrLoop.Name
      Next usrLoop

      ' Delete the new Group object because this
      ' is a demonstration.
      .Groups.Delete "NewGroup"

   End With

End Sub
```

CreateUser Method and Password and PID Properties Example

This example uses the **CreateUser** method and **Password** and **PID** properties to create a new **User** object; it then makes the new **User** object a member of different **Group** objects and lists its properties and groups.

```
Sub CreateUserX()

   Dim wrkDefault As Workspace
   Dim usrNew As User
   Dim grpNew As Group
   Dim usrTemp As User
   Dim prpLoop As Property
   Dim grpLoop As Group

   Set wrkDefault = DBEngine.Workspaces(0)

   With wrkDefault

      ' Create and append new User.
      Set usrNew = .CreateUser("NewUser")
      usrNew.PID = "AAA123456789"
      usrNew.Password = "NewPassword"
      .Users.Append usrNew

      ' Create and append new Group.
      Set grpNew = .CreateGroup("NewGroup", _
         "AAA123456789")
      .Groups.Append grpNew

      ' Make the user "NewUser" a member of the
      ' group "NewGroup" by creating and adding the
      ' appropriate User object to the group's Users
      ' collection.
      Set usrTemp = _
         .Groups("NewGroup").CreateUser("NewUser")
      .Groups("NewGroup").Users.Append usrTemp

      Debug.Print "Properties of " & usrNew.Name

      ' Enumerate the Properties collection of NewUser. The
      ' PID property is not readable.
      For Each prpLoop In usrNew.Properties
         On Error Resume Next
         If prpLoop <> "" Then Debug.Print "    " & _
            prpLoop.Name & " = " & prpLoop
         On Error GoTo 0
      Next prpLoop

      Debug.Print "Groups collection of " & usrNew.Name

      ' Enumerate the Groups collection of NewUser.
      For Each grpLoop In usrNew.Groups
         Debug.Print "    " & _
            grpLoop.Name
      Next grpLoop
```

```
      ' Delete the new User and Group objects because this
      ' is a demonstration.
      .Users.Delete "NewUser"
      .Groups.Delete "NewGroup"

   End With

End Sub
```

CreateWorkspace Method Example

This example uses the **CreateWorkspace** method to create both a Microsoft Jet workspace and an ODBCDirect workspace. It then lists the properties of both types of workspace.

```
Sub CreateWorkspaceX()

   Dim wrkODBC As Workspace
   Dim wrkJet As Workspace
   Dim wrkLoop As Workspace
   Dim prpLoop As Property

   ' Create an ODBCDirect workspace. Until you create
   ' Microsoft Jet workspace, the Microsoft Jet database
   ' engine will not be loaded into memory.
   Set wrkODBC = CreateWorkspace("ODBCWorkspace", "admin", _
      "", dbUseODBC)
   Workspaces.Append wrkODBC

   DefaultType = dbUseJet
   ' Create an unnamed Workspace object of the type
   ' specified by the DefaultType property of DBEngine
   ' (dbUseJet).
   Set wrkJet = CreateWorkspace("", "admin", "")

   ' Enumerate Workspaces collection.
   Debug.Print "Workspace objects in Workspaces collection:"
   For Each wrkLoop In Workspaces
      Debug.Print "    " & wrkLoop.Name
   Next wrkLoop

   With wrkODBC
      ' Enumerate Properties collection of ODBCDirect
      ' workspace.
      Debug.Print "Properties of " & .Name
      On Error Resume Next
      For Each prpLoop In .Properties
         Debug.Print "    " & prpLoop.Name & " = " & prpLoop
      Next prpLoop
      On Error GoTo 0
   End With

   With wrkJet
      ' Enumerate Properties collection of Microsoft Jet
      ' workspace.
      Debug.Print _
         "Properties of unnamed Microsoft Jet workspace"
      On Error Resume Next
      For Each prpLoop In .Properties
         Debug.Print "    " & prpLoop.Name & " = " & prpLoop
      Next prpLoop
      On Error GoTo 0
   End With

   wrkODBC.Close
   wrkJet.Close
```

```
End Sub
```

Idle Method Example

This example uses the **Idle** method to ensure that an output procedure is accessing the most current data available from the database. The IdleOutput procedure is required for this procedure to run.

```
Sub IdleX()

   Dim dbsNorthwind As Database
   Dim strCountry As String
   Dim strSQL As String
   Dim rstOrders As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Get name of country from user and build SQL statement
   ' with it.
   strCountry = Trim(InputBox("Enter country:"))
   strSQL = "SELECT * FROM Orders WHERE ShipCountry = '" & _
      strCountry & "' ORDER BY OrderID"

   ' Open Recordset object with SQL statement.
   Set rstOrders = dbsNorthwind.OpenRecordset(strSQL)

   ' Display contents of Recordset object.
   IdleOutput rstOrders, strCountry

   rstOrders.Close
   dbsNorthwind.Close

End Sub

Sub IdleOutput(rstTemp As Recordset, strTemp As String)

   ' Call the Idle method to release unneeded locks, force
   ' pending writes, and refresh the memory with the current
   ' data in the .mdb file.
   DBEngine.Idle dbRefreshCache

   ' Enumerate the Recordset object.
   With rstTemp
      Debug.Print "Orders from " & strTemp & ":"
      Debug.Print , "OrderID", "CustomerID", "OrderDate"
      Do While Not .EOF
         Debug.Print , !OrderID, !CustomerID, !OrderDate
         .MoveNext
      Loop
   End With

End Sub
```

MakeReplica Method Example

This function uses the **MakeReplica** method to create an additional replica of an existing Design Master. The `intOptions` argument can be a combination of the constants **dbRepMakeReadOnly** and **dbRepMakePartial**, or it can be 0. For example, to create a read-only partial replica, you should pass the value `dbRepMakeReadOnly + dbRepMakePartial` as the value of `intOptions`.

```
Function MakeAdditionalReplica(strReplicableDB As _
   String, strNewReplica As String, intOptions As _
   Integer) As Integer

   Dim dbsTemp As Database
   On Error GoTo ErrorHandler

   Set dbsTemp = OpenDatabase(strReplicableDB)

   ' If no options are passed to
   ' MakeAdditionalReplica, omit the
   ' options argument, which defaults to
   ' a full, read/write replica. Otherwise,
   ' use the value of intOptions.

   If intOptions = 0 Then
      dbsTemp.MakeReplica strNewReplica, _
         "Replica of " & strReplicableDB
   Else
      dbsTemp.MakeReplica strNewReplica, _
         "Replica of " & strReplicableDB, _
         intOptions
   End If

   dbsTemp.Close

ErrorHandler:
   Select Case Err
      Case 0:
         MakeAdditionalReplica = 0
         Exit Function
      Case Else:
         MsgBox "Error " & Err & " : " & Error
         MakeAdditionalReplica = Err
         Exit Function
   End Select

End Function
```

NextRecordset Method and DefaultCursorDriver Property Example

This example uses the **NextRecordset** method to view the data from a compound SELECT query. The **DefaultCursorDriver** property must be set to **dbUseODBCCursor** when executing such queries. The **NextRecordset** method will return **True** even if some or all of the SELECT statements return zero records – it will return **False** only after all the individual SQL clauses have been checked.

```
Sub NextRecordsetX()

   Dim wrkODBC As Workspace
   Dim conPubs As Connection
   Dim rstTemp As Recordset
   Dim intCount As Integer
   Dim booNext As Boolean

   ' Create ODBCDirect Workspace object and open Connection
   ' object. The DefaultCursorDriver setting is required
   ' when using compound SQL statements.
   Set wrkODBC = CreateWorkspace("", _
      "admin", "", dbUseODBC)
   wrkODBC.DefaultCursorDriver = dbUseODBCCursor
   Set conPubs = wrkODBC.OpenConnection("Publishers", , , _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")

   ' Construct compound SELECT statement.
   Set rstTemp = conPubs.OpenRecordset("SELECT * " & _
      "FROM authors; " & _
      "SELECT * FROM stores; " & _
      "SELECT * FROM jobs")

   ' Try printing results from each of the three SELECT
   ' statements.
   booNext = True
   intCount = 1
   With rstTemp
      Do While booNext
         Debug.Print "Contents of recordset #" & intCount
         Do While Not .EOF
            Debug.Print , .Fields(0), .Fields(1)
            .MoveNext
         Loop
         booNext = .NextRecordset
         Debug.Print "    rstTemp.NextRecordset = " & _
            booNext
         intCount = intCount + 1
      Loop
   End With

   rstTemp.Close
   conPubs.Close
   wrkODBC.Close

End Sub
```

Another way to accomplish the same task would be to create a prepared statement containing the compound SQL statement. The **CacheSize** property of the **QueryDef** object must be set to 1, and the **Recordset** object must be forward-only and read-only.

```
Sub NextRecordsetX2()

   Dim wrkODBC As Workspace
   Dim conPubs As Connection
   Dim qdfTemp As QueryDef
   Dim rstTemp As Recordset
   Dim intCount As Integer
   Dim booNext As Boolean

   ' Create ODBCDirect Workspace object and open Connection
   ' object. The DefaultCursorDriver setting is required
   ' when using compound SQL statements.
   Set wrkODBC = CreateWorkspace("", _
      "admin", "", dbUseODBC)
   wrkODBC.DefaultCursorDriver = dbUseODBCCursor
   Set conPubs = wrkODBC.OpenConnection("Publishers", , , _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")

   ' Create a temporary stored procedure with a compound
   ' SELECT statement.
   Set qdfTemp = conPubs.CreateQueryDef("", _
      "SELECT * FROM authors; " & _
      "SELECT * FROM stores; " & _
      "SELECT * FROM jobs")
   ' Set CacheSize and open Recordset object with arguments
   ' that will allow access to multiple recordsets.
   qdfTemp.CacheSize = 1
   Set rstTemp = qdfTemp.OpenRecordset(dbOpenForwardOnly, _
      dbReadOnly)

   ' Try printing results from each of the three SELECT
   ' statements.
   booNext = True
   intCount = 1
   With rstTemp
      Do While booNext
         Debug.Print "Contents of recordset #" & intCount
         Do While Not .EOF
            Debug.Print , .Fields(0), .Fields(1)
            .MoveNext
         Loop
         booNext = .NextRecordset
         Debug.Print "    rstTemp.NextRecordset = " & _
            booNext
         intCount = intCount + 1
      Loop
   End With

   rstTemp.Close
   qdfTemp.Close
   conPubs.Close
   wrkODBC.Close

End Sub
```

OpenConnection Method Example

This example uses the **OpenConnection** method with different parameters to open three different **Connection** objects.

```
Sub OpenConnectionX()

   Dim wrkODBC As Workspace
   Dim conPubs As Connection
   Dim conPubs2 As Connection
   Dim conPubs3 As Connection
   Dim conLoop As Connection

   ' Create ODBCDirect Workspace object.
   Set wrkODBC = CreateWorkspace("NewODBCWorkspace", _
      "admin", "", dbUseODBC)

   ' Open Connection object using supplied information in
   ' the connect string. If this information were
   ' insufficient, you could trap for an error rather than
   ' go to an ODBC Driver Manager dialog box.
   MsgBox "Opening Connection1..."
   Set conPubs = wrkODBC.OpenConnection("Connection1", _
      dbDriverNoPrompt, , _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")

   ' Open read-only Connection object based on information
   ' you enter in the ODBC Driver Manager dialog box.
   MsgBox "Opening Connection2..."
   Set conPubs2 = wrkODBC.OpenConnection("Connection2", _
      dbDriverPrompt, True, "ODBC;DSN=Publishers;")

   ' Open read-only Connection object by entering only the
   ' missing information in the ODBC Driver Manager dialog
   ' box.
   MsgBox "Opening Connection3..."
   Set conPubs3 = wrkODBC.OpenConnection("Connection3", _
      dbDriverCompleteRequired, True, _
      "ODBC;DATABASE=pubs;DSN=Publishers;")

   ' Enumerate the Connections collection.
   For Each conLoop In wrkODBC.Connections
      Debug.Print "Connection properties for " & _
         conLoop.Name & ":"

      With conLoop
         ' Print property values by explicitly calling each
         ' Property object; the Connection object does not
         ' support a Properties collection.
         Debug.Print "    Connect = " & .Connect
         ' Property actually returns a Database object.
         Debug.Print "    Database[.Name] = " & _
            .Database.Name
         Debug.Print "    Name = " & .Name
         Debug.Print "    QueryTimeout = " & .QueryTimeout
         Debug.Print "    RecordsAffected = " & _
            .RecordsAffected
```

```
         Debug.Print "    StillExecuting = " & _
            .StillExecuting
         Debug.Print "    Transactions = " & .Transactions
         Debug.Print "    Updatable = " & .Updatable
      End With

   Next conLoop

   conPubs.Close
   conPubs2.Close
   conPubs3.Close
   wrkODBC.Close

End Sub
```

OpenDatabase Method Example

This example uses the **OpenDatabase** method to open one Microsoft Jet database and two Microsoft Jet-connected ODBC databases.

```
Sub OpenDatabaseX()

   Dim wrkJet As Workspace
   Dim dbsNorthwind As Database
   Dim dbsPubs As Database
   Dim dbsPubs2 As Database
   Dim dbsLoop As Database
   Dim prpLoop As Property

   ' Create Microsoft Jet Workspace object.
   Set wrkJet = CreateWorkspace("", "admin", "", dbUseJet)

   ' Open Database object from saved Microsoft Jet database
   ' for exclusive use.
   MsgBox "Opening Northwind..."
   Set dbsNorthwind = wrkJet.OpenDatabase("Northwind.mdb", _
      True)

   ' Open read-only Database object based on information in
   ' the connect string.
   MsgBox "Opening pubs..."
   Set dbsPubs = wrkJet.OpenDatabase("Publishers", _
      dbDriverNoPrompt, True, _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")

   ' Open read-only Database object by entering only the
   ' missing information in the ODBC Driver Manager dialog
   ' box.
   MsgBox "Opening second copy of pubs..."
   Set dbsPubs2 = wrkJet.OpenDatabase("Publishers", _
      dbDriverCompleteRequired, True, _
      "ODBC;DATABASE=pubs;DSN=Publishers;")

   ' Enumerate the Databases collection.
   For Each dbsLoop In wrkJet.Databases
      Debug.Print "Database properties for " & _
         dbsLoop.Name & ":"

      On Error Resume Next
      ' Enumerate the Properties collection of each Database
      ' object.
      For Each prpLoop In dbsLoop.Properties
         If prpLoop.Name = "Connection" Then
            ' Property actually returns a Connection object.
            Debug.Print "    Connection[.Name] = " & _
               dbsLoop.Connection.Name
         Else
            Debug.Print "    " & prpLoop.Name & " = " & _
               prpLoop
         End If
      Next prpLoop
      On Error GoTo 0
```

```
    Next dbsLoop

    dbsNorthwind.Close
    dbsPubs.Close
    dbsPubs2.Close
    wrkJet.Close

End Sub
```

OpenRecordset Method Example

This example uses the **OpenRecordset** method to open five different **Recordset** objects and display their contents. The OpenRecordsetOutput procedure is required for this procedure to run.

```
Sub OpenRecordsetX()

   Dim wrkJet As Workspace
   Dim wrkODBC As Workspace
   Dim dbsNorthwind As Database
   Dim conPubs As Connection
   Dim rstTemp As Recordset
   Dim rstTemp2 As Recordset

   ' Open Microsoft Jet and ODBCDirect workspaces, Microsoft
   ' Jet database, and ODBCDirect connection.
   Set wrkJet = CreateWorkspace("", "admin", "", dbUseJet)
   Set wrkODBC = CreateWorkspace("", "admin", "", dbUseODBC)
   Set dbsNorthwind = wrkJet.OpenDatabase("Northwind.mdb")
   Set conPubs = wrkODBC.OpenConnection("", , , _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")

   ' Open five different Recordset objects and display the
   ' contents of each.

   Debug.Print "Opening forward-only-type recordset " & _
      "where the source is a QueryDef object..."
   Set rstTemp = dbsNorthwind.OpenRecordset( _
      "Ten Most Expensive Products", dbOpenForwardOnly)
   OpenRecordsetOutput rstTemp

   Debug.Print "Opening read-only dynaset-type " & _
      "recordset where the source is an SQL statement..."
   Set rstTemp = dbsNorthwind.OpenRecordset( _
      "SELECT * FROM Employees", dbOpenDynaset, dbReadOnly)
   OpenRecordsetOutput rstTemp

   ' Use the Filter property to retrieve only certain
   ' records with the next OpenRecordset call.
   Debug.Print "Opening recordset from existing " & _
      "Recordset object to filter records..."
   rstTemp.Filter = "LastName >= 'M'"
   Set rstTemp2 = rstTemp.OpenRecordset()
   OpenRecordsetOutput rstTemp2

   Debug.Print "Opening dynamic-type recordset from " & _
      "an ODBC connection..."
   Set rstTemp = conPubs.OpenRecordset( _
      "SELECT * FROM stores", dbOpenDynamic)
   OpenRecordsetOutput rstTemp

   ' Use the StillExecuting property to determine when the
   ' Recordset is ready for manipulation.
   Debug.Print "Opening snapshot-type recordset based " & _
      "on asynchronous query to ODBC connection..."
   Set rstTemp = conPubs.OpenRecordset("publishers", _
      dbOpenSnapshot, dbRunAsync)
```

```
   Do While rstTemp.StillExecuting
      Debug.Print "    [still executing...]"
   Loop
   OpenRecordsetOutput rstTemp

   rstTemp.Close
   dbsNorthwind.Close
   conPubs.Close
   wrkJet.Close
   wrkODBC.Close

End Sub

Sub OpenRecordsetOutput(rstOutput As Recordset)

   ' Enumerate the specified Recordset object.
   With rstOutput
      Do While Not .EOF
         Debug.Print , .Fields(0), .Fields(1)
         .MoveNext
      Loop
   End With

End Sub
```

PopulatePartial Method Example

The following example uses the **PopulatePartial** method after changing a replica filter.

```
Sub PopulatePartialX()

   Dim tdfCustomers As TableDef
   Dim strFilter As String
   Dim dbsTemp As Database

   ' Open the partial replica in exclusive mode.
   Set dbsTemp = OpenDatabase("F:\SALES\FY96CA.MDB", True)

   With dbsTemp
      Set tdfCustomers = .TableDefs("Customers")

      ' Synchronize with full replica
      ' before setting replica filter.
      .Synchronize "C:\SALES\FY96.MDB"

      strFilter = "Region = 'CA'"
      tdfCustomers.ReplicaFilter = strFilter

      ' Populate records from the full replica.
      .PopulatePartial "C:\SALES\FY96.MDB"

      .Close
   End With

End Sub
```

RefreshLink Method Example

This example uses the **RefreshLink** method to refresh the data in a linked table after its connection has been changed from one data source to another. The RefreshLinkOutput procedure is required for this procedure to run.

```
Sub RefreshLinkX()

   Dim dbsCurrent As Database
   Dim tdfLinked As TableDef

   ' Open a database to which a linked table can be
   ' appended.
   Set dbsCurrent = OpenDatabase("DB1.mdb")

   ' Create a linked table that points to a Microsoft
   ' SQL Server database.
   Set tdfLinked = _
      dbsCurrent.CreateTableDef("AuthorsTable")
   tdfLinked.Connect = _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers"
   tdfLinked.SourceTableName = "authors"
   dbsCurrent.TableDefs.Append tdfLinked

   ' Display contents of linked table.
   Debug.Print _
      "Data from linked table connected to first source:"
   RefreshLinkOutput dbsCurrent

   ' Change connection information for linked table and
   ' refresh the connection in order to make the new data
   ' available.
   tdfLinked.Connect = _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=NewPublishers"
   tdfLinked.RefreshLink

   ' Display contents of linked table.
   Debug.Print _
      "Data from linked table connected to second source:"
   RefreshLinkOutput dbsCurrent

   ' Delete linked table because this is a demonstration.
   dbsCurrent.TableDefs.Delete tdfLinked.Name

   dbsCurrent.Close

End Sub

Sub RefreshLinkOutput(dbsTemp As Database)

   Dim rstRemote As Recordset
   Dim intCount As Integer

   ' Open linked table.
   Set rstRemote = _
      dbsTemp.OpenRecordset("AuthorsTable")
```

```
    intCount = 0

    ' Enumerate Recordset object, but stop at 50 records.
    With rstRemote
        Do While Not .EOF And intCount < 50
            Debug.Print , .Fields(0), .Fields(1)
            intCount = intCount + 1
            .MoveNext
        Loop
        If Not .EOF Then Debug.Print , "[more records]"
        .Close
    End With

End Sub
```

RegisterDatabase Method Example

This example uses the **RegisterDatabase** method to register a Microsoft SQL Server data source named Publishers in the Windows Registry.

Using the Windows ODBC Control Panel icon is the preferred way to create, modify, or delete data source names.

```
Sub RegisterDatabaseX()

   Dim dbsRegister As Database
   Dim strDescription As String
   Dim strAttributes As String
   Dim errLoop As Error

   ' Build keywords string.
   strDescription = InputBox( "Enter a description " & _
      "for the database to be registered.")
   strAttributes = "Database=pubs" & _
      vbCr & "Description=" & strDescription & _
      vbCr & "OemToAnsi=No" & _
      vbCr & "Server=Server1"

   ' Update Windows Registry.
   On Error GoTo Err_Register
   DBEngine.RegisterDatabase "Publishers", "SQL Server", _
      True, strAttributes
   On Error GoTo 0

   MsgBox "Use regedit.exe to view changes: " & _
      "HKEY_CURRENT_USER\" & _
      "Software\ODBC\ODBC.INI"

   Exit Sub

Err_Register:

   ' Notify user of any errors that result from
   ' the invalid data.
   If DBEngine.Errors.Count > 0 Then
      For Each errLoop In DBEngine.Errors
         MsgBox "Error number: " & errLoop.Number & _
            vbCr & errLoop.Description
      Next errLoop
   End If

   Resume Next

End Sub
```

Requery Method Example

This example shows how the **Requery** method can be used to refresh a query after underlying data has been changed.

```
Sub RequeryX()

   Dim dbsNorthwind As Database
   Dim qdfTemp As QueryDef
   Dim rstView As Recordset
   Dim rstChange As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set qdfTemp = dbsNorthwind.CreateQueryDef("", _
      "PARAMETERS ViewCountry Text; " & _
      "SELECT FirstName, LastName, Country FROM " & _
      "Employees WHERE Country = [ViewCountry] " & _
      "ORDER BY LastName")

   qdfTemp.Parameters!ViewCountry = "USA"
   Debug.Print "Data after initial query, " & _
      [ViewCountry] = USA"
   Set rstView = qdfTemp.OpenRecordset
   Do While Not rstView.EOF
      Debug.Print "     " & rstView!FirstName & " " & _
         rstView!LastName & ", " & rstView!Country
      rstView.MoveNext
   Loop

   ' Change underlying data.
   Set rstChange = dbsNorthwind.OpenRecordset("Employees")
   rstChange.AddNew
   rstChange!FirstName = "Nina"
   rstChange!LastName = "Roberts"
   rstChange!Country = "USA"
   rstChange.Update

   rstView.Requery
   Debug.Print "Requery after changing underlying data"
   Set rstView = qdfTemp.OpenRecordset
   Do While Not rstView.EOF
      Debug.Print "     " & rstView!FirstName & " " & _
         rstView!LastName & ", " & rstView!Country
      rstView.MoveNext
   Loop

   ' Restore original data because this is only a
   ' demonstration.
   rstChange.Bookmark = rstChange.LastModified
   rstChange.Delete
   rstChange.Close

   rstView.Close
   dbsNorthwind.Close

End Sub
```

This example shows how the **Requery** method can be used to refresh a query after the query parameters have been changed.

```
Sub RequeryX2()

   Dim dbsNorthwind As Database
   Dim qdfTemp As QueryDef
   Dim prmCountry As Parameter
   Dim rstView As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set qdfTemp = dbsNorthwind.CreateQueryDef("", _
      "PARAMETERS ViewCountry Text; " & _
      "SELECT FirstName, LastName, Country FROM " & _
      "Employees WHERE Country = [ViewCountry] " & _
      "ORDER BY LastName")
   Set prmCountry = qdfTemp.Parameters!ViewCountry

   qdfTemp.Parameters!ViewCountry = "USA"
   Debug.Print "Data after initial query, " & _
      [ViewCountry] = USA"
   Set rstView = qdfTemp.OpenRecordset
   Do While Not rstView.EOF
      Debug.Print "    " & rstView!FirstName & " " & _
         rstView!LastName & ", " & rstView!Country
      rstView.MoveNext
   Loop

   ' Change query parameter.
   qdfTemp.Parameters!ViewCountry = "UK"
   ' QueryDef argument must be included so that the
   ' resulting Recordset reflects the change in the query
   ' parameter.
   rstView.Requery qdfTemp
   Debug.Print "Requery after changing parameter, " & _
      "[ViewCountry] = UK"
   Do While Not rstView.EOF
      Debug.Print "    " & rstView!FirstName & " " & _
         rstView!LastName & ", " & rstView!Country
      rstView.MoveNext
   Loop

   rstView.Close
   dbsNorthwind.Close

End Sub
```

SetOption Method Example

This example uses the **SetOption** method to change the value of two registry keys based on input from the user. The **SetOption** method only overrides the stored registry values for the current application. The stored settings will remain unchanged and will be the only values visible to the user through REGEDIT.EXE.

```
Sub SetOptionX()

   Dim intExclusiveDelay As Integer
   Dim intSharedDelay As Integer

   ' Get user input for new values of ExclusiveAsyncDelay
   ' and SharedAsyncDelay registry keys.
   intExclusiveDelay = Val(InputBox("Enter a new value " & _
      " for the ExclusiveAsyncDelay registry key " & _
      "(in milliseconds):"))
   intSharedDelay = Val(InputBox("Enter a new value " & _
      "for the SharedAsyncDelay registry key " & _
      "(in milliseconds):"))

   If intExclusiveDelay > 0 And intSharedDelay > 0 Then
      ' Change values of registry keys.
      SetOption dbExclusiveAsyncDelay, intExclusiveDelay
      SetOption dbSharedAsyncDelay, intSharedDelay
      MsgBox "Registry keys changed to new values " & _
         "for duration of program."
   Else
      MsgBox "Registry keys left unchanged."
   End If

End Sub
```

Synchronize Method Example

These four examples use the **Synchronize** method to demonstrate one-way and bi-directional exchanges of information between two members of a replica set. They will work if you have converted Northwind.mdb to a Design Master (see the **Replicable** Property), and created a replica from it. The replica name specified is Nwreplica.mdb. Change the name of the replica to fit your situation, or use the **MakeReplica** method to create a replica if you need one.

This example sends the changes from the Northwind Design Master to Nwreplica. Adjust the paths to the locations of the files on your computer.

```
Sub SendChangeToReplicaX()

   Dim dbsNorthwind As Database

   ' Opens the replicable database Northwind.mdb.
   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Sends data or structural changes to the replica.
   dbsNorthwind.Synchronize "Nwreplica.mdb", _
      dbRepExportChanges

   dbsNorthwind.Close

End Sub
```

In this example, the replicable database Northwind.mdb receives changes from the replica in the path — Nwreplica. You must run this procedure from the database receiving the changes.

```
Sub ReceiveChangeX()

   Dim dbsNorthwind As Database

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Sends changes from replica to Design Master.
   dbsNorthwind.Synchronize "Nwreplica.mdb", _
      dbRepImportChanges

   dbsNorthwind.Close

End Sub
```

In this example, changes from both the replicable database Northwind and a replica are exchanged. This is the default argument for this method.

```
Sub TwoWayExchangeX()

   Dim dbsNorthwind As Database

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Sends changes made in each replica to the other.
   dbsNorthwind.Synchronize "Nwreplica.mdb", _
      dbRepImpExpChanges

   dbsNorthwind.Close

End Sub
```

The following code sample synchronizes two databases over the Internet.

```
Sub InternetSynchronizeX()

   Dim dbsTemp As Database

   Set dbsTemp = OpenDatabase("C:\Data\OrdEntry.mdb")

   ' Synchronize the local database with the replica on
   ' the Internet server.
   dbsTemp.Synchronize _
      "www.mycompany.myserver.com" _
      & "/files/Orders.mdb", _
      dbRepImpExpChanges + dbRepSyncInternet

   dbsTemp.Close

End Sub
```

AbsolutePosition Property Example

This example uses the **AbsolutePosition** property to track the progress of a loop that enumerates all the records of a **Recordset**.

```
Sub AbsolutePositionX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim strMessage As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   ' AbsolutePosition only works with dynasets or snapshots.
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees", _
      dbOpenSnapshot)

   With rstEmployees
      ' Populate Recordset.
      .MoveLast
      .MoveFirst

      ' Enumerate Recordset.
      Do While Not .EOF
         ' Display current record information. Add 1 to
         ' AbsolutePosition value because it is zero-based.
         strMessage = "Employee: " & !LastName & vbCr & _
            "(record " & (.AbsolutePosition + 1) & _
            " of " & .RecordCount & ")"
         If MsgBox(strMessage, vbOKCancel) = vbCancel _
            Then Exit Do
         .MoveNext
      Loop

      .Close
   End With

   dbsNorthwind.Close

End Sub
```

AllowZeroLength Property Example

In this example, the **AllowZeroLength** property allows the user to set the value of a **Field** to an empty string. In this situation, the user can distinguish between a record where data is not known and a record where the data does not apply.

```
Sub AllowZeroLengthX()

   Dim dbsNorthwind As Database
   Dim tdfEmployees As TableDef
   Dim fldTemp As Field
   Dim rstEmployees As Recordset
   Dim strMessage As String
   Dim strInput As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set tdfEmployees = dbsNorthwind.TableDefs("Employees")
   ' Create a new Field object and append it to the Fields
   ' collection of the Employees table.
   Set fldTemp = tdfEmployees.CreateField("FaxPhone", _
      dbText, 24)
   fldTemp.AllowZeroLength = True
   tdfEmployees.Fields.Append fldTemp

   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")

   With rstEmployees
      ' Get user input.
      .Edit
      strMessage = "Enter fax number for " & _
         !FirstName & " " & !LastName & "." & vbCr & _
         "[? - unknown, X - has no fax]"
      strInput = UCase(InputBox(strMessage))
      If strInput <> "" Then
         Select Case strInput
            Case "?"
               !FaxPhone = Null
            Case "X"
               !FaxPhone = ""
            Case Else
               !FaxPhone = strInput
         End Select

         .Update

         ' Print report.
         Debug.Print "Name - Fax number"
         Debug.Print !FirstName & " " & !LastName & " - ";

         If IsNull(!FaxPhone) Then
            Debug.Print "[Unknown]"
         Else
            If !FaxPhone = "" Then
               Debug.Print "[Has no fax]"
            Else
               Debug.Print !FaxPhone
```

```
            End If
         End If

      Else
         .CancelUpdate
      End If

      .Close
   End With

   ' Delete new field because this is a demonstration.
   tdfEmployees.Fields.Delete fldTemp.Name
   dbsNorthwind.Close

End Sub
```

BOF, EOF Properties Example

This example demonstrates how the **BOF** and **EOF** properties let the user move forward and backward through a **Recordset**.

```
Sub BOFX()

   Dim dbsNorthwind As Database
   Dim rstCategories As Recordset
   Dim strMessage As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstCategories = _
      dbsNorthwind.OpenRecordset("Categories", _
      dbOpenSnapshot)

   With rstCategories
      ' Populate Recordset.
      .MoveLast
      .MoveFirst

      Do While True
         ' Display current record information and get user
         ' input.
         strMessage = "Category: " & !CategoryName & _
            vbCr & "(record " & (.AbsolutePosition + 1) & _
            " of " & .RecordCount & ")" & vbCr & vbCr & _
            "Enter 1 to go forward, 2 to go backward:"

         ' Move forward or backward and trap for BOF or EOF.
         Select Case InputBox(strMessage)
            Case 1
               .MoveNext
               If .EOF Then
                  MsgBox _
                     "End of the file!" & vbCr & _
                     "Pointer being moved to last record."
                  .MoveLast
               End If

            Case 2
               .MovePrevious
               If .BOF Then
                  MsgBox _
                     "Beginning of the file!" & vbCr & _
                     "Pointer being moved to first record."
                  .MoveFirst
               End If

            Case Else
               Exit Do
         End Select

      Loop

      .Close
   End With
```

```
    dbsNorthwind.Close

End Sub
```

Bookmark and Bookmarkable Properties Example

This example uses the **Bookmark** and **Bookmarkable** properties to let the user flag a record in a **Recordset** and return to it later.

```
Sub BookmarkX()

   Dim dbsNorthwind As Database
   Dim rstCategories As Recordset
   Dim strMessage As String
   Dim intCommand As Integer
   Dim varBookmark As Variant

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstCategories = _
      dbsNorthwind.OpenRecordset("Categories", _
      dbOpenSnapshot)

   With rstCategories

      If .Bookmarkable = False Then
         Debug.Print "Recordset is not Bookmarkable!"
      Else
         ' Populate Recordset.
         .MoveLast
         .MoveFirst

         Do While True
            ' Show information about current record and get
            ' user input.
            strMessage = "Category: " & !CategoryName & _
               " (record " & (.AbsolutePosition + 1) & _
               " of " & .RecordCount & ")" & vbCr & _
               "Enter command:" & vbCr & _
               "[1 - next / 2 - previous /" & vbCr & _
               "3 - set bookmark / 4 - go to bookmark]"
            intCommand = Val(InputBox(strMessage))

            Select Case intCommand
               ' Move forward or backward, trapping for BOF
               ' or EOF.
               Case 1
                  .MoveNext
                  If .EOF Then .MoveLast
               Case 2
                  .MovePrevious
                  If .BOF Then .MoveFirst

               ' Store the bookmark of the current record.
               Case 3
                  varBookmark = .Bookmark

               ' Go to the record indicated by the stored
               ' bookmark.
               Case 4
                  If IsEmpty(varBookmark) Then
                     MsgBox "No Bookmark set!"
```

```
                  Else
                     .Bookmark = varBookmark
                  End If

               Case Else
                  Exit Do
            End Select

         Loop

      End If

      .Close
   End With

   dbsNorthwind.Close

End Sub
```

Count Property Example

This example demonstrates the **Count** property with three different collections in the Northwind 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.

```
Sub CountX()

   Dim dbsNorthwind As Database
   Dim intloop As Integer

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      ' Print information about TableDefs collection.
      Debug.Print .TableDefs.Count & _
         " TableDefs in Northwind"
      For intloop = 0 To .TableDefs.Count - 1
         Debug.Print "    " & .TableDefs(intloop).Name
      Next intloop

      ' Print information about QueryDefs collection.
      Debug.Print .QueryDefs.Count & _
         " QueryDefs in Northwind"
      For intloop = 0 To .QueryDefs.Count - 1
         Debug.Print "    " & .QueryDefs(intloop).Name
      Next intloop

      ' Print information about Relations collection.
      Debug.Print .Relations.Count & _
         " Relations in Northwind"
      For intloop = 0 To .Relations.Count - 1
         Debug.Print "    " & .Relations(intloop).Name
      Next intloop

      .Close
   End With

End Sub
```

DataUpdatable Property Example

This example demonstrates the **DataUpdatable** property using the first field from six different **Recordsets**. The DataOutput function is required for this procedure to run.

```
Sub DataUpdatableX()

   Dim dbsNorthwind As Database
   Dim rstNorthwind As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      ' Open and print report about a table-type Recordset.
      Set rstNorthwind = .OpenRecordset("Employees")
      DataOutput rstNorthwind

      ' Open and print report about a dynaset-type Recordset.
      Set rstNorthwind = .OpenRecordset("Employees", _
         dbOpenDynaset)
      DataOutput rstNorthwind

      ' Open and print report about a snapshot-type Recordset.
      Set rstNorthwind = .OpenRecordset("Employees", _
         dbOpenSnapshot)
      DataOutput rstNorthwind

      ' Open and print report about a forward-only-type Recordset.
      Set rstNorthwind = .OpenRecordset("Employees", _
         dbOpenForwardOnly)
      DataOutput rstNorthwind

      ' Open and print report about a Recordset based on
      ' a select query.
      Set rstNorthwind = _
         .OpenRecordset("Current Product List")
      DataOutput rstNorthwind

      ' Open and print report about a Recordset based on a
      ' select query that calculates totals.
      Set rstNorthwind = .OpenRecordset("Order Subtotals")
      DataOutput rstNorthwind

      .Close
   End With

End Sub

Function DataOutput(rstTemp As Recordset)

   With rstTemp
      Debug.Print "Recordset: " & .Name & ", ";
      Select Case .Type
         Case dbOpenTable
            Debug.Print "dbOpenTable"
         Case dbOpenDynaset
            Debug.Print "dbOpenDynaset"
```

```
         Case dbOpenSnapshot
            Debug.Print "dbOpenSnapshot"
         Case dbOpenForwardOnly
            Debug.Print "dbOpenForwardOnly"
      End Select
      Debug.Print "    Field: " & .Fields(0).Name & ", " & _
         "DataUpdatable = " & .Fields(0).DataUpdatable
      Debug.Print
      .Close
   End With

End Function
```

DateCreated, LastUpdated Properties Example

This example demonstrates the **DateCreated** and **LastUpdated** properties by adding a new **Field** to an existing **TableDef** and by creating a new **TableDef**. The DateOutput function is required for this procedure to run.

```
Sub DateCreatedX()

   Dim dbsNorthwind As Database
   Dim tdfEmployees As TableDef
   Dim tdfNewTable As TableDef

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      Set tdfEmployees = .TableDefs!Employees

      With tdfEmployees
         ' Print current information about the Employees
         ' table.
         DateOutput "Current properties", tdfEmployees

         ' Create and append a field to the Employees table.
         .Fields.Append .CreateField("NewField", dbDate)

         ' Print new information about the Employees
         ' table.
         DateOutput "After creating a new field", _
            tdfEmployees

         ' Delete new Field because this is a demonstration.
         .Fields.Delete "NewField"
      End With

      ' Create and append a new TableDef object to the
      ' Northwind database.
      Set tdfNewTable = .CreateTableDef("NewTableDef")
      With tdfNewTable
         .Fields.Append .CreateField("NewField", dbDate)
      End With
      .TableDefs.Append tdfNewTable

      ' Print information about the new TableDef object.
      DateOutput "After creating a new table", tdfNewTable

      ' Delete new TableDef object because this is a
      ' demonstration.
      .TableDefs.Delete tdfNewTable.Name
      .Close
   End With

End Sub

Function DateOutput(strTemp As String, _
   tdfTemp As TableDef)

   ' Print DateCreated and LastUpdated information about
```

```
    ' specified TableDef object.
    Debug.Print strTemp
    Debug.Print "    TableDef: " & tdfTemp.Name
    Debug.Print "        DateCreated = " & _
       tdfTemp.DateCreated
    Debug.Print "        LastUpdated = " & _
       tdfTemp.LastUpdated
    Debug.Print

End Function
```

DefaultValue Property Example

This example uses the **DefaultValue** property to alert the user of a field's normal value while prompting for input. In addition, it demonstrates how new records will be filled using **DefaultValue** in the absence of any other input. The DefaultPrompt function is required for this procedure to run.

```
Sub DefaultValueX()

   Dim dbsNorthwind As Database
   Dim tdfEmployees As TableDef
   Dim strOldDefault As String
   Dim rstEmployees As Recordset
   Dim strMessage As String
   Dim strCode As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set tdfEmployees = dbsNorthwind.TableDefs!Employees

   ' Store original DefaultValue information and set the
   ' property to a new value.
   strOldDefault = _
      tdfEmployees.Fields!PostalCode.DefaultValue
   tdfEmployees.Fields!PostalCode.DefaultValue = "98052"

   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees", _
      dbOpenDynaset)

   With rstEmployees
      ' Add a new record to the Recordset.
      .AddNew
      !FirstName = "Bruce"
      !LastName = "Oberg"

      ' Get user input. If user enters something, the field
      ' will be filled with that data; otherwise, it will be
      ' filled with the DefaultValue information.
      strMessage = "Enter postal code for " & vbCr & _
         !FirstName & " " & !LastName & ":"
      strCode = DefaultPrompt(strMessage, !PostalCode)
      If strCode <> "" Then !PostalCode = strCode
      .Update

      ' Go to new record and print information.
      .Bookmark = .LastModified
      Debug.Print "    FirstName = " & !FirstName
      Debug.Print "    LastName = " & !LastName
      Debug.Print "    PostalCode = " & !PostalCode

      ' Delete new record because this is a demonstration.
      .Delete
      .Close
   End With

   ' Restore original DefaultValue property because this is a
   ' demonstration.
   tdfEmployees.Fields!PostalCode.DefaultValue = _
```

```
      strOldDefault

   dbsNorthwind.Close

End Sub

Function DefaultPrompt(strPrompt As String, _
   fldTemp As Field) As String

   Dim strFullPrompt As String

   ' Ask user for new DefaultValue setting for the specified
   ' Field object.
   strFullPrompt = strPrompt & vbCr & _
      "[Default = " & fldTemp.DefaultValue & _
      ", Cancel - use default]"
   DefaultPrompt = InputBox(strFullPrompt)

End Function
```

DistinctCount Property Example

This example uses the **DistinctCount** property to show how you can determine the number of unique values in an **Index** object. However, this value is only accurate immediately after creating the **Index**. It will remain accurate if no keys change, or if new keys are added and no old keys are deleted; otherwise, it will not be reliable. (If this procedure is run several times, you can see the effect on the **DistinctCount** property values of the existing **Index** objects.)

```
Sub DistinctCountX()

   Dim dbsNorthwind As Database
   Dim tdfEmployees As TableDef
   Dim idxCountry As Index
   Dim idxLoop As Index
   Dim rstEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set tdfEmployees = dbsNorthwind!Employees

   With tdfEmployees
      ' Create and append new Index object to the Employees
      ' table.
      Set idxCountry = .CreateIndex("CountryIndex")
      idxCountry.Fields.Append _
         idxCountry.CreateField("Country")
      .Indexes.Append idxCountry

      ' The collection must be refreshed for the new
      ' DistinctCount data to be available.
      .Indexes.Refresh

      ' Enumerate Indexes collection to show the current
      ' DistinctCount values.
      Debug.Print "Indexes before adding new record"
      For Each idxLoop In .Indexes
         Debug.Print "    DistinctCount = " & _
            idxLoop.DistinctCount & ", Name = " & _
            idxLoop.Name
      Next idxLoop

      Set rstEmployees = _
         dbsNorthwind.OpenRecordset("Employees")

      ' Add a new record to the Employees table.
      With rstEmployees
         .AddNew
         !FirstName = "April"
         !LastName = "LaMonte"
         !Country = "Canada"
         .Update
      End With

      ' Enumerate Indexes collection to show the modified
      ' DistinctCount values.
      Debug.Print "Indexes after adding new record and " & _
         "refreshing Indexes"
```

```
      .Indexes.Refresh
      For Each idxLoop In .Indexes
         Debug.Print "    DistinctCount = " & _
            idxLoop.DistinctCount & ", Name = " & _
            idxLoop.Name
      Next idxLoop

      ' Delete new record because this is a demonstration.
      With rstEmployees
         .Bookmark = .LastModified
         .Delete
         .Close
      End With

      ' Delete new Indexes because this is a demonstration.
      .Indexes.Delete idxCountry.Name
   End With

   dbsNorthwind.Close

End Sub
```

EditMode Property Example

This example shows the value of the **EditMode** property under various conditions. The EditModeOutput function is required for this procedure to run.

```
Sub EditModeX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees", _
      dbOpenDynaset)

   ' Show the EditMode property under different editing
   ' states.
   With rstEmployees
      EditModeOutput "Before any Edit or AddNew:", .EditMode
      .Edit
      EditModeOutput "After Edit:", .EditMode
      .Update
      EditModeOutput "After Update:", .EditMode
      .AddNew
      EditModeOutput "After AddNew:", .EditMode
      .CancelUpdate
      EditModeOutput "After CancelUpdate:", .EditMode
      .Close
   End With

   dbsNorthwind.Close

End Sub

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 dbEditNone
         Debug.Print "dbEditNone"
      Case dbEditInProgress
         Debug.Print "dbEditInProgress"
      Case dbEditAdd
         Debug.Print "dbEditAdd"
   End Select

End Function
```

FieldSize Property Example

This example uses the **FieldSize** property to list the number of bytes used by the Memo and Long Binary **Field** objects in two different tables.

```
Sub FieldSizeX()

   Dim dbsNorthwind As Database
   Dim rstCategories As Recordset
   Dim rstEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstCategories = _
      dbsNorthwind.OpenRecordset("Categories", _
      dbOpenDynaset)
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees", _
      dbOpenDynaset)

   Debug.Print _
      "Field sizes from records in Categories table"

   With rstCategories
      Debug.Print "    CategoryName - " & _
         "Description (bytes) - Picture (bytes)"

   ' Enumerate the Categories Recordset and print the size
   ' in bytes of the picture field for each record.
      Do While Not .EOF
         Debug.Print "        " & !CategoryName & " - " & _
            !Description.FieldSize & " - " & _
            !Picture.FieldSize
         .MoveNext
      Loop

      .Close
   End With

   Debug.Print "Field sizes from records in Employees table"

   With rstEmployees
      Debug.Print "    LastName - Notes (bytes) - " & _
         "Photo (bytes)"

   ' Enumerate the Employees Recordset and print the size
   ' in bytes of the picture field for each record.
      Do While Not .EOF
         Debug.Print "        " & !LastName & " - " & _
            !Notes.FieldSize & " - " & !Photo.FieldSize
         .MoveNext
      Loop

      .Close
   End With

   dbsNorthwind.Close
```

```
End Sub
```

Filter Property Example

This example uses the **Filter** property to create a new **Recordset** from an existing **Recordset** based on a specified condition. The FilterField function is required for this procedure to run.

```
Sub FilterX()

   Dim dbsNorthwind As Database
   Dim rstOrders As Recordset
   Dim intOrders As Integer
   Dim strCountry As String
   Dim rstOrdersCountry As Recordset
   Dim strMessage As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstOrders = dbsNorthwind.OpenRecordset("Orders", _
      dbOpenSnapshot)

   ' Populate the Recordset.
   rstOrders.MoveLast
   intOrders = rstOrders.RecordCount

   ' Get user input.
   strCountry = Trim(InputBox( _
      "Enter a country to filter on:"))

   If strCountry <> "" Then
      ' Open a filtered Recordset object.
      Set rstOrdersCountry = _
         FilterField(rstOrders, "ShipCountry", strCountry)

      With rstOrdersCountry
         ' Check RecordCount before populating Recordset;
         ' otherwise, error may result.
         If .RecordCount <> 0 Then .MoveLast
         ' Print number of records for the original
         ' Recordset object and the filtered Recordset
         ' object.
         strMessage = "Orders in original recordset: " & _
            vbCr & intOrders & vbCr & _
            "Orders in filtered recordset (Country = '" & _
            strCountry & "'): " & vbCr & .RecordCount
         MsgBox strMessage
         .Close
      End With

   End If

   rstOrders.Close

   dbsNorthwind.Close

End Sub

Function FilterField(rstTemp As Recordset, _
   strField As String, strFilter As String) As Recordset
```

```
    ' Set a filter on the specified Recordset object and then
    ' open a new Recordset object.
    rstTemp.Filter = strField & " = '" & strFilter & "'"
    Set FilterField = rstTemp.OpenRecordset

End Function
```

Note To see the effects of filtering `rstOrders`, you must set its **Filter** property, and then open a second **Recordset** object based on `rstOrders`.

Note When you know the data you want to select, it's usually more efficient to create a **Recordset** with an SQL statement. This example shows how you can create just one **Recordset** and obtain records from a particular country.

```
Sub FilterX2()

    Dim dbsNorthwind As Database
    Dim rstOrders As Recordset

    Set dbsNorthwind = OpenDatabase("Northwind.mdb")

    ' Open a Recordset object that selects records from a
    ' table based on the shipping country.
    Set rstOrders = _
       dbsNorthwind.OpenRecordset("SELECT * " & _
       "FROM Orders WHERE ShipCountry = 'USA'", _
       dbOpenSnapshot)

    rstOrders.Close
    dbsNorthwind.Close

End Sub
```

Foreign Property Example

This example shows how the **Foreign** property can indicate which **Index** objects in a **TableDef** are foreign key indexes. Such indexes are created by the Microsoft Jet database engine when a **Relation** is created. The default name for the foreign key indexes is the name of the primary table plus the name of the foreign table. The ForeignOutput function is required for this procedure to run.

```
Sub ForeignX()

   Dim dbsNorthwind As Database

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      ' Print report on foreign key indexes from two
      ' TableDef objects and a QueryDef object.
      ForeignOutput .TableDefs!Products
      ForeignOutput .TableDefs!Orders
      ForeignOutput .TableDefs![Order Details]

      .Close
   End With

End Sub

Function ForeignOutput(tdfTemp As TableDef)

   Dim idxLoop As Index

   With tdfTemp
      Debug.Print "Indexes in " & .Name & " TableDef"
      ' Enumerate the Indexes collection of the specified
      ' TableDef object.
      For Each idxLoop In .Indexes
         Debug.Print "    " & idxLoop.Name
         Debug.Print "        Foreign = " & idxLoop.Foreign
      Next idxLoop
   End With

End Function
```

ForeignName, ForeignTable, and Table Properties Example

This example shows how the **Table**, **ForeignTable**, and **ForeignName** properties define the terms of a **Relation** between two tables.

```
Sub ForeignNameX()

   Dim dbsNorthwind As Database
   Dim relLoop As Relation

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   Debug.Print "Relation"
   Debug.Print "                    Table - Field"
   Debug.Print "    Primary (One)    ";
   Debug.Print ".Table - .Fields(0).Name"
   Debug.Print "    Foreign (Many)   ";
   Debug.Print ".ForeignTable - .Fields(0).ForeignName"

   ' Enumerate the Relations collection of the Northwind
   ' database to report on the property values of
   ' the Relation objects and their Field objects.
   For Each relLoop In dbsNorthwind.Relations
      With relLoop
         Debug.Print
         Debug.Print .Name & " Relation"
         Debug.Print "                    Table - Field"
         Debug.Print "    Primary (One)    ";
         Debug.Print .Table & " - " & .Fields(0).Name
         Debug.Print "    Foreign (Many)   ";
         Debug.Print .ForeignTable & " - " & _
            .Fields(0).ForeignName
      End With
   Next relLoop

   dbsNorthwind.Close

End Sub
```

IgnoreNulls Property Example

This example sets the **IgnoreNulls** property of a new **Index** to **True** or **False** based on user input, and then demonstrates the effect on a **Recordset** with a record whose key field contains a **Null** value.

```
Sub IgnoreNullsX()

   Dim dbsNorthwind As Database
   Dim tdfEmployees As TableDef
   Dim idxNew As Index
   Dim rstEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set tdfEmployees = dbsNorthwind!Employees

   With tdfEmployees
      ' Create a new Index object.
      Set idxNew = .CreateIndex("NewIndex")
      idxNew.Fields.Append idxNew.CreateField("Country")

      ' Set the IgnoreNulls property of the new Index object
      ' based on the user's input.
      Select Case MsgBox("Set IgnoreNulls to True?", _
         vbYesNoCancel)
         Case vbYes
            idxNew.IgnoreNulls = True
         Case vbNo
            idxNew.IgnoreNulls = False
         Case Else
            dbsNorthwind.Close
            End
      End Select

      ' Append the new Index object to the Indexes
      ' collection of the Employees table.
      .Indexes.Append idxNew
      .Indexes.Refresh
   End With

   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")

   With rstEmployees
      ' Add a new record to the Employees table.
      .AddNew
      !FirstName = "Gary"
      !LastName = "Haarsager"
      .Update

      ' Use the new index to set the order of the records.
      .Index = idxNew.Name
      .MoveFirst

      Debug.Print "Index = " & .Index & _
         ", IgnoreNulls = " & idxNew.IgnoreNulls
      Debug.Print "    Country - Name"
```

```
      ' Enumerate the Recordset. The value of the
      ' IgnoreNulls property will determine if the newly
      ' added record appears in the output.
      Do While Not .EOF
         Debug.Print "        " & _
            IIf(IsNull(!Country), "[Null]", !Country) & _
            " - " & !FirstName & " " & !LastName
         .MoveNext
      Loop

      ' Delete new record because this is a demonstration.
      .Index = ""
      .Bookmark = .LastModified
      .Delete
      .Close
   End With

   ' Delete new Index because this is a demonstration.
   tdfEmployees.Indexes.Delete idxNew.Name
   dbsNorthwind.Close

End Sub
```

Index Property Example

This example uses the **Index** property to set different record orders for a table-type **Recordset**.

```
Sub IndexPropertyX()

   Dim dbsNorthwind As Database
   Dim tdfEmployees As TableDef
   Dim rstEmployees As Recordset
   Dim idxLoop As Index

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")
   Set tdfEmployees = dbsNorthwind.TableDefs!Employees

   With rstEmployees

      ' Enumerate Indexes collection of Employees table.
      For Each idxLoop In tdfEmployees.Indexes
         .Index = idxLoop.Name
         Debug.Print "Index = " & .Index
         Debug.Print "    EmployeeID - PostalCode - Name"
         .MoveFirst

         ' Enumerate Recordset to show the order of records.
         Do While Not .EOF
            Debug.Print "        " & !EmployeeID & " - " & _
               !PostalCode & " - " & !FirstName & " " & _
               !LastName
            .MoveNext
         Loop

      Next idxLoop

      .Close
   End With

   dbsNorthwind.Close

End Sub
```

LastModified Property Example

This example uses the **LastModified** property to move the current record pointer to both a record that has been modified and a newly created record.

```
Sub LastModifiedX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim strFirst As String
   Dim strLast As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees", _
      dbOpenDynaset)

   With rstEmployees
      ' Store current data.
      strFirst = !FirstName
      strLast = !LastName
      ' Change data in current record.
      .Edit
      !FirstName = "Julie"
      !LastName = "Warren"
      .Update
      ' Move current record pointer to the most recently
      ' changed or added record.
      .Bookmark = .LastModified
      Debug.Print _
         "Data in LastModified record after Edit: " & _
         !FirstName & " " & !LastName

      ' Restore original data because this is a demonstration.
      .Edit
      !FirstName = strFirst
      !LastName = strLast
      .Update

      ' Add new record.
      .AddNew
      !FirstName = "Roger"
      !LastName = "Harui"
      .Update
      ' Move current record pointer to the most recently
      ' changed or added record.
      .Bookmark = .LastModified
      Debug.Print _
         "Data in LastModified record after AddNew: " & _
         !FirstName & " " & !LastName

      ' Delete new record because this is a demonstration.
      .Delete
      .Close
   End With

   dbsNorthwind.Close
```

```
End Sub
```

LockEdits Property Example

This example demonstrates pessimistic locking by setting the **LockEdits** property to **True**, and then demonstrates optimistic locking by setting the **LockEdits** property to **False**. It also demonstrates what kind of error handling is required in a multiuser database environment in order to modify a field. The PessimisticLock and OptimisticLock functions are required for this procedure to run.

```
Sub LockEditsX()

   Dim dbsNorthwind As Database
   Dim rstCustomers As Recordset
   Dim strOldName As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstCustomers = _
      dbsNorthwind.OpenRecordset("Customers", _
      dbOpenDynaset)

   With rstCustomers
      ' Store original data.
      strOldName = !CompanyName

      If MsgBox("Pessimistic locking demonstration...", _
            vbOKCancel) = vbOK Then

         ' Attempt to modify data with pessimistic locking
         ' in effect.
         If PessimisticLock(rstCustomers, !CompanyName, _
               "Acme Foods") Then
            MsgBox "Record successfully edited."

            ' Restore original data...
            .Edit
            !CompanyName = strOldName
            .Update
         End If

      End If

      If MsgBox("Optimistic locking demonstration...", _
            vbOKCancel) = vbOK Then

         ' Attempt to modify data with optimistic locking
         ' in effect.
         If OptimisticLock(rstCustomers, !CompanyName, _
               "Acme Foods") Then
            MsgBox "Record successfully edited."

            ' Restore original data...
            .Edit
            !CompanyName = strOldName
            .Update
         End If

      End If

      .Close
```

```
   End With

   dbsNorthwind.Close

End Sub

Function PessimisticLock(rstTemp As Recordset, _
   fldTemp As Field, strNew As String) As Boolean

   dim ErrLoop as Error

   PessimisticLock = True

   With rstTemp
      .LockEdits = True

      ' When you set LockEdits to True, you trap for errors
      ' when you call the Edit method.
      On Error GoTo Err_Lock
      .Edit
      On Error GoTo 0

      ' If the Edit is still in progress, then no errors
      ' were triggered; you may modify the data.
      If .EditMode = dbEditInProgress Then
         fldTemp = strNew
         .Update
         .Bookmark = .LastModified
      Else
         ' Retrieve current record to see changes made by
         ' other user.
         .Move 0
      End If

   End With

   Exit Function

Err_Lock:

   If DBEngine.Errors.Count > 0 Then
      ' Enumerate the Errors collection.
      For Each errLoop In DBEngine.Errors
         MsgBox "Error number: " & errLoop.Number & _
            vbCr & errLoop.Description
      Next errLoop
      PessimisticLock = False
   End If

   Resume Next

End Function

Function OptimisticLock(rstTemp As Recordset, _
   fldTemp As Field, strNew As String) As Boolean

   dim ErrLoop as Error
```

```
   OptimisticLock = True

   With rstTemp
      .LockEdits = False
      .Edit
      fldTemp = strNew

      ' When you set LockEdits to False, you trap for errors
      ' when you call the Update method.
      On Error GoTo Err_Lock
      .Update
      On Error GoTo 0

      ' If there is no Edit in progress, then no errors were
      ' triggered; you may modify the data.
      If .EditMode = dbEditNone Then
         ' Move current record pointer to the most recently
         ' modified record.
         .Bookmark = .LastModified
      Else
         .CancelUpdate
         ' Retrieve current record to see changes made by
         ' other user.
         .Move 0
      End If

   End With

   Exit Function

Err_Lock:

   If DBEngine.Errors.Count > 0 Then
      ' Enumerate the Errors collection.
      For Each errLoop In DBEngine.Errors
         MsgBox "Error number: " & errLoop.Number & _
            vbCr & errLoop.Description
      Next errLoop
      OptimisticLock = False
   End If

   Resume Next

End Function
```

NoMatch Property Example

This example uses the **NoMatch** property to determine whether a **Seek** and a **FindFirst** were successful, and if not, to give appropriate feedback. The SeekMatch and FindMatch procedures are required for this procedure to run.

```
Sub NoMatchX()

   Dim dbsNorthwind As Database
   Dim rstProducts As Recordset
   Dim rstCustomers As Recordset
   Dim strMessage As String
   Dim strSeek As String
   Dim strCountry As String
   Dim varBookmark As Variant

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   ' Default is dbOpenTable; required if Index property will
   ' be used.
   Set rstProducts = dbsNorthwind.OpenRecordset("Products")

   With rstProducts
      .Index = "PrimaryKey"

      Do While True
         ' Show current record information; ask user for
         ' input.
         strMessage = "NoMatch with Seek method" & vbCr & _
            "Product ID: " & !ProductID & vbCr & _
            "Product Name: " & !ProductName & vbCr & _
            "NoMatch = " & .NoMatch & vbCr & vbCr & _
            "Enter a product ID."
         strSeek = InputBox(strMessage)
         If strSeek = "" Then Exit Do

         ' Call procedure that seeks for a record based on
         ' the ID number supplied by the user.
         SeekMatch rstProducts, Val(strSeek)
      Loop

      .Close
   End With

   Set rstCustomers = dbsNorthwind.OpenRecordset( _
      "SELECT CompanyName, Country FROM Customers " & _
      "ORDER BY CompanyName", dbOpenSnapshot)

   With rstCustomers

      Do While True
         ' Show current record information; ask user for
         ' input.
         strMessage = "NoMatch with FindFirst method" & _
            vbCr & "Customer Name: " & !CompanyName & _
            vbCr & "Country: " & !Country & vbCr & _
            "NoMatch = " & .NoMatch & vbCr & vbCr & _
            "Enter country on which to search."
```

```
         strCountry = Trim(InputBox(strMessage))
         If strCountry = "" Then Exit Do

         ' Call procedure that finds a record based on
         ' the country name supplied by the user.
         FindMatch rstCustomers, _
            "Country = '" & strCountry & "'"
      Loop

      .Close
   End With

   dbsNorthwind.Close

End Sub

Sub SeekMatch(rstTemp As Recordset, _
   intSeek As Integer)

   Dim varBookmark As Variant
   Dim strMessage As String

   With rstTemp
      ' Store current record location.
      varBookmark = .Bookmark
      .Seek "=", intSeek

      ' If Seek method fails, notify user and return to the
      ' last current record.
      If .NoMatch Then
         strMessage = _
            "Not found! Returning to current record." & _
            vbCr & vbCr & "NoMatch = " & .NoMatch
         MsgBox strMessage
         .Bookmark = varBookmark
      End If

   End With

End Sub

Sub FindMatch(rstTemp As Recordset, _
   strFind As String)

   Dim varBookmark As Variant
   Dim strMessage As String

   With rstTemp
      ' Store current record location.
      varBookmark = .Bookmark
      .FindFirst strFind

      ' If Find method fails, notify user and return to the
      ' last current record.
      If .NoMatch Then
         strMessage = _
            "Not found! Returning to current record." & _
```

```
            vbCr & vbCr & "NoMatch = " & .NoMatch
        MsgBox strMessage
        .Bookmark = varBookmark
      End If

    End With

End Sub
```

OrdinalPosition Property Example

This example changes the **OrdinalPosition** property values in the Employees **TableDef** in order to control the **Field** order in a resulting **Recordset**. By setting the **OrdinalPosition** of all the **Fields** to 1, any resulting **Recordset** will order the **Fields** alphabetically. Note that the **OrdinalPosition** values in the **Recordset** don't match the values in the **TableDef**, but simply reflect the end result of the **TableDef** changes.

```
Sub OrdinalPositionX()

   Dim dbsNorthwind As Database
   Dim tdfEmployees As TableDef
   Dim aintPosition() As Integer
   Dim astrFieldName() As String
   Dim intTemp As Integer
   Dim fldTemp As Field
   Dim rstEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set tdfEmployees = dbsNorthwind.TableDefs("Employees")

   With tdfEmployees
      ' Display and store original OrdinalPosition data.
      Debug.Print _
         "Original OrdinalPosition data in TableDef."
      ReDim aintPosition(0 To .Fields.Count - 1) As Integer
      ReDim astrFieldName(0 To .Fields.Count - 1) As String
      For intTemp = 0 To .Fields.Count - 1
         aintPosition(intTemp) = _
            .Fields(intTemp).OrdinalPosition
         astrFieldName(intTemp) = .Fields(intTemp).Name
         Debug.Print , aintPosition(intTemp), _
            astrFieldName(intTemp)
      Next intTemp

      ' Change OrdinalPosition data.
      For Each fldTemp In .Fields
         fldTemp.OrdinalPosition = 1
      Next fldTemp

      ' Open new Recordset object to show how the
      ' OrdinalPosition data has affected the record order.
      Debug.Print _
         "OrdinalPosition data from resulting Recordset."
      Set rstEmployees = dbsNorthwind.OpenRecordset( _
         "SELECT * FROM Employees")
      For Each fldTemp In rstEmployees.Fields
         Debug.Print , fldTemp.OrdinalPosition, fldTemp.Name
      Next fldTemp
      rstEmployees.Close

      ' Restore original OrdinalPosition data because this is
      ' a demonstration.
      For intTemp = 0 To .Fields.Count - 1
         .Fields(astrFieldName(intTemp)).OrdinalPosition = _
            aintPosition(intTemp)
```

```
        Next intTemp
    End With
    dbsNorthwind.Close
End Sub
```

PercentPosition Property Example

This example uses the **PercentPosition** property to show the position of the current record pointer relative to the beginning of the **Recordset**.

```
Sub PercentPositionX()

   Dim dbsNorthwind As Database
   Dim rstProducts As Recordset
   Dim strFind As String
   Dim strMessage As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   ' PercentPosition only works with dynasets or snapshots.
   Set rstProducts = dbsNorthwind.OpenRecordset( _
      "SELECT ProductName FROM Products " & _
      "ORDER BY ProductName", dbOpenSnapshot)

   With rstProducts
      ' Populate the Recordset.
      .MoveLast
      .MoveFirst

      Do While True
         ' Show current record information and ask user
         ' for input.
         strMessage = "Product: " & !ProductName & vbCr & _
            "The record pointer is " & _
            Format(.PercentPosition, "##0.0") & _
            "% from the " & vbCr & _
            "beginning of the Recordset." & vbCr & _
            "Please enter a character search string " & _
            "for a product name."
         strFind = Trim(InputBox(strMessage))
         If strFind = "" Then Exit Do

         ' Try to find a record matching the search string.
         .FindFirst "ProductName >= '" & strFind & "'"
         If .NoMatch Then .MoveLast
      Loop

      .Close
   End With

   dbsNorthwind.Close

End Sub
```

Primary Property Example

This example uses the **Primary** property to designate a new **Index** in a new **TableDef** as the primary **Index** for that table. Note that setting the **Primary** property to **True** automatically sets **Unique** and **Required** properties to **True** as well.

```
Sub PrimaryX()

   Dim dbsNorthwind As Database
   Dim tdfNew As TableDef
   Dim idxNew As Index
   Dim idxLoop As Index
   Dim fldLoop As Field
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Create and append a new TableDef object to the
   ' TableDefs collection of the Northwind database.
   Set tdfNew = dbsNorthwind.CreateTableDef("NewTable")
   tdfNew.Fields.Append tdfNew.CreateField("NumField", _
      dbLong, 20)
   tdfNew.Fields.Append tdfNew.CreateField("TextField", _
      dbText, 20)
   dbsNorthwind.TableDefs.Append tdfNew

   With tdfNew
      ' Create and append a new Index object to the
      ' Indexes collection of the new TableDef object.
      Set idxNew = .CreateIndex("NumIndex")
      idxNew.Fields.Append idxNew.CreateField("NumField")
      idxNew.Primary = True
      .Indexes.Append idxNew
      Set idxNew = .CreateIndex("TextIndex")
      idxNew.Fields.Append idxNew.CreateField("TextField")
      .Indexes.Append idxNew

      Debug.Print .Indexes.Count & " Indexes in " & _
         .Name & " TableDef"

      ' Enumerate Indexes collection.
      For Each idxLoop In .Indexes

         With idxLoop
            Debug.Print "    " & .Name

            ' Enumerate Fields collection of each Index
            ' object.
            Debug.Print "        Fields"
            For Each fldLoop In .Fields
               Debug.Print "            " & fldLoop.Name
            Next fldLoop

            ' Enumerate Properties collection of each
            ' Index object.
            Debug.Print "        Properties"
            For Each prpLoop In .Properties
```

```
            Debug.Print "            " & prpLoop.Name & _
               " = " & IIf(prpLoop = "", "[empty]", _
               prpLoop)
         Next prpLoop
      End With

   Next idxLoop

End With

dbsNorthwind.TableDefs.Delete tdfNew.Name
dbsNorthwind.Close

End Sub
```

RecordCount Property Example

This example demonstrates the **RecordCount** property with different types of **Recordsets** before and after they're populated.

```
Sub RecordCountX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      ' Open table-type Recordset and show RecordCount
      ' property.
      Set rstEmployees = .OpenRecordset("Employees")
      Debug.Print _
         "Table-type recordset from Employees table"
      Debug.Print "    RecordCount = " & _
         rstEmployees.RecordCount
      rstEmployees.Close

      ' Open dynaset-type Recordset and show RecordCount
      ' property before populating the Recordset.
      Set rstEmployees = .OpenRecordset("Employees", _
         dbOpenDynaset)
      Debug.Print "Dynaset-type recordset " & _
         "from Employees table before MoveLast"
      Debug.Print "    RecordCount = " & _
         rstEmployees.RecordCount

      ' Show the RecordCount property after populating the
      ' Recordset.
      rstEmployees.MoveLast
      Debug.Print "Dynaset-type recordset " & _
         "from Employees table after MoveLast"
      Debug.Print "    RecordCount = " & _
         rstEmployees.RecordCount
      rstEmployees.Close

      ' Open snapshot-type Recordset and show RecordCount
      ' property before populating the Recordset.
      Set rstEmployees = .OpenRecordset("Employees", _
         dbOpenSnapshot)
      Debug.Print "Snapshot-type recordset " & _
         "from Employees table before MoveLast"
      Debug.Print "    RecordCount = " & _
         rstEmployees.RecordCount

      ' Show the RecordCount property after populating the
      ' Recordset.
      rstEmployees.MoveLast
      Debug.Print "Snapshot-type recordset " & _
         "from Employees table after MoveLast"
      Debug.Print "    RecordCount = " & _
         rstEmployees.RecordCount
      rstEmployees.Close
```

```
      ' Open forward-only-type Recordset and show
      ' RecordCount property before populating the
      ' Recordset.
      Set rstEmployees = .OpenRecordset("Employees", _
         dbOpenForwardOnly)
      Debug.Print "Forward-only-type recordset " & _
         "from Employees table before MoveLast"
      Debug.Print "    RecordCount = " & _
         rstEmployees.RecordCount

      ' Show the RecordCount property after calling the
      ' MoveNext method.
      rstEmployees.MoveNext
      Debug.Print "Forward-only-type recordset " & _
         "from Employees table after MoveNext"
      Debug.Print "    RecordCount = " & _
         rstEmployees.RecordCount
      rstEmployees.Close

      .Close
   End With

End Sub
```

RecordsAffected Property Example

This example uses the **RecordsAffected** property with action queries executed from a **Database** object and from a **QueryDef** object. The RecordsAffectedOutput function is required for this procedure to run.

```
Sub RecordsAffectedX()

   Dim dbsNorthwind As Database
   Dim qdfTemp As QueryDef
   Dim strSQLChange As String
   Dim strSQLRestore As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      ' Print report of contents of the Employees
      ' table.
      Debug.Print _
         "Number of records in Employees table: " & _
         .TableDefs!Employees.RecordCount
      RecordsAffectedOutput dbsNorthwind

      ' Define and execute an action query.
      strSQLChange = "UPDATE Employees " & _
         "SET Country = 'United States' " & _
         "WHERE Country = 'USA'"
      .Execute strSQLChange

      ' Print report of contents of the Employees
      ' table.
      Debug.Print _
         "RecordsAffected after executing query " & _
         "from Database: " & .RecordsAffected
      RecordsAffectedOutput dbsNorthwind

      ' Define and run another action query.
      strSQLRestore = "UPDATE Employees " & _
         "SET Country = 'USA' " & _
         "WHERE Country = 'United States'"
      Set qdfTemp = .CreateQueryDef("", strSQLRestore)
      qdfTemp.Execute

      ' Print report of contents of the Employees
      ' table.
      Debug.Print _
         "RecordsAffected after executing query " & _
         "from QueryDef: " & qdfTemp.RecordsAffected
      RecordsAffectedOutput dbsNorthwind

      .Close

   End With

End Sub

Function RecordsAffectedOutput(dbsNorthwind As Database)
```

```
   Dim rstEmployees As Recordset

   ' Open a Recordset object from the Employees table.
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")

   With rstEmployees
      ' Enumerate Recordset.
      .MoveFirst
      Do While Not .EOF
         Debug.Print "    " & !LastName & ", " & !Country
         .MoveNext
      Loop
      .Close
   End With

End Function
```

Restartable Property Example

This example demonstrates the **Restartable** property with different **Recordset** objects.

```
Sub RestartableX()

   Dim dbsNorthwind As Database
   Dim rstTemp As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      ' Open a table-type Recordset and print its
      ' Restartable property.
      Set rstTemp = .OpenRecordset("Employees", dbOpenTable)
      Debug.Print _
         "Table-type recordset from Employees table"
      Debug.Print "    Restartable = " & rstTemp.Restartable
      rstTemp.Close

      ' Open a Recordset from an SQL statement and print its
      ' Restartable property.
      Set rstTemp = _
         .OpenRecordset("SELECT * FROM Employees")
      Debug.Print "Recordset based on SQL statement"
      Debug.Print "    Restartable = " & rstTemp.Restartable
      rstTemp.Close

      ' Open a Recordset from a saved QueryDef object and
      ' print its Restartable property.
      Set rstTemp = .OpenRecordset("Current Product List")
      Debug.Print _
         "Recordset based on permanent QueryDef (" & _
         rstTemp.Name & ")"
      Debug.Print "    Restartable = " & rstTemp.Restartable
      rstTemp.Close

      .Close
   End With

End Sub
```

Size Property Example

This example demonstrates the **Size** property by enumerating the names and sizes of the **Field** objects in the Employees table.

```
Sub SizeX()

   Dim dbsNorthwind As Database
   Dim tdfEmployees As TableDef
   Dim fldNew As Field
   Dim fldLoop As Field

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set tdfEmployees = dbsNorthwind.TableDefs!Employees

   With tdfEmployees

      ' Create and append a new Field object to the
      ' Employees table.
      Set fldNew = .CreateField("FaxPhone")
      fldNew.Type = dbText
      fldNew.Size = 20
      .Fields.Append fldNew

      Debug.Print "TableDef: " & .Name
      Debug.Print "    Field.Name - Field.Type - Field.Size"

      ' Enumerate Fields collection; print field names,
      ' types, and sizes.
      For Each fldLoop In .Fields
         Debug.Print "        " & fldLoop.Name & " - " & _
            fldLoop.Type & " - " & fldLoop.Size
      Next fldLoop

      ' Delete new field because this is a demonstration.
      .Fields.Delete fldNew.Name

   End With

   dbsNorthwind.Close

End Sub
```

Sort Property Example

This example demonstrates the **Sort** property by changing its value and creating a new **Recordset**. The SortOutput function is required for this procedure to run.

```
Sub SortX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim rstSortEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees", _
      dbOpenDynaset)

   With rstEmployees
      SortOutput "Original Recordset:", rstEmployees
      .Sort = "LastName, FirstName"
      ' Print report showing Sort property and record order.
      SortOutput _
         "Recordset after changing Sort property:", _
         rstEmployees
      ' Open new Recordset from current one.
      Set rstSortEmployees = .OpenRecordset
      ' Print report showing Sort property and record order.
      SortOutput "New Recordset:", rstSortEmployees
      rstSortEmployees.Close
      .Close
   End With

   dbsNorthwind.Close

End Sub

Function SortOutput(strTemp As String, _
   rstTemp As Recordset)

   With rstTemp
      Debug.Print strTemp
      Debug.Print "    Sort = " & _
         IIf(.Sort <> "", .Sort, "[Empty]")
      .MoveFirst

      ' Enumerate Recordset.
      Do While Not .EOF
         Debug.Print "         " & !LastName & _
            ", " & !FirstName
         .MoveNext
      Loop

   End With

End Function
```

Note When you know the data you want to select, it's usually more efficient to create a **Recordset** with an SQL statement. This example shows how you can create just one **Recordset** and obtain the

same results as in the preceding example.

```
Sub SortX2()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   ' Open a Recordset from an SQL statement that specifies a
   ' sort order.
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("SELECT * " & _
      "FROM Employees ORDER BY LastName, FirstName", _
      dbOpenDynaset)

   dbsNorthwind.Close

End Sub
```

SourceField, SourceTable Properties Example

This example demonstrates the **SourceField** and **SourceTable** properties by opening a **Recordset** made up of fields from two tables.

```
Sub SourceFieldX()

   Dim dbsNorthwind As Database
   Dim rstProductCategory As Recordset
   Dim fldLoop As Field
   Dim strSQL As String

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   ' Open a Recordset from an SQL statement that uses fields
   ' from two different tables.
   strSQL = "SELECT ProductID AS ProdID, " & _
      "ProductName AS ProdName, " & _
      "Categories.CategoryID AS CatID, " & _
      "CategoryName AS CatName " & _
      "FROM Categories INNER JOIN Products ON " & _
      "Categories.CategoryID = Products.CategoryID " & _
      "ORDER BY ProductName"
   Set rstProductCategory = _
      dbsNorthwind.OpenRecordset(strSQL)

   Debug.Print "Field - SourceTable - SourceField"
   ' Enumerate Fields collection of Recordset, printing
   ' name, original table, and original name.
   For Each fldLoop In rstProductCategory.Fields
      Debug.Print "    " & fldLoop.Name & " - " & _
         fldLoop.SourceTable & " - " & fldLoop.SourceField
   Next fldLoop

   rstProductCategory.Close
   dbsNorthwind.Close

End Sub
```

SQL Property Example

This example demonstrates the **SQL** property by setting and changing the **SQL** property of a temporary **QueryDef** and comparing the results. The SQLOutput function is required for this procedure to run.

```
Sub SQLX()

   Dim dbsNorthwind As Database
   Dim qdfTemp As QueryDef
   Dim rstEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set qdfTemp = dbsNorthwind.CreateQueryDef("")

   ' Open Recordset using temporary QueryDef object and
   ' print report.
   SQLOutput "SELECT * FROM Employees " & _
      "WHERE Country = 'USA' " & _
      "ORDER BY LastName", qdfTemp

   ' Open Recordset using temporary QueryDef object and
   ' print report.
   SQLOutput "SELECT * FROM Employees " & _
      "WHERE Country = 'UK' " & _
      "ORDER BY LastName", qdfTemp

   dbsNorthwind.Close

End Sub

Function SQLOutput(strSQL As String, qdfTemp As QueryDef)

   Dim rstEmployees As Recordset

   ' Set SQL property of temporary QueryDef object and open
   ' a Recordset.
   qdfTemp.SQL = strSQL
   Set rstEmployees = qdfTemp.OpenRecordset

   Debug.Print strSQL

   With rstEmployees
      ' Enumerate Recordset.
      Do While Not .EOF
         Debug.Print "     " & !FirstName & " " & _
            !LastName & ", " & !Country
         .MoveNext
      Loop
      .Close
   End With

End Function
```

Type Property Example

This example demonstrates the **Type** property by returning the name of the constant corresponding to the value of the **Type** property of four different **Recordsets**. The RecordsetType function is required for this procedure to run.

```
Sub TypeX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Default is dbOpenTable.
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")
   Debug.Print _
      "Table-type recordset (Employees table): " & _
      RecordsetType(rstEmployees.Type)
   rstEmployees.Close

   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees", _
      dbOpenDynaset)
   Debug.Print _
      "Dynaset-type recordset (Employees table): " & _
      RecordsetType(rstEmployees.Type)
   rstEmployees.Close

   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees", _
      dbOpenSnapshot)
   Debug.Print _
      "Snapshot-type recordset (Employees table): " & _
      RecordsetType(rstEmployees.Type)
   rstEmployees.Close

   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees", _
      dbOpenForwardOnly)
   Debug.Print _
      "Forward-only-type recordset (Employees table): " & _
      RecordsetType(rstEmployees.Type)
   rstEmployees.Close

   dbsNorthwind.Close

End Sub

Function RecordsetType(intType As Integer) As String

   Select Case intType
      Case dbOpenTable
         RecordsetType = "dbOpenTable"
      Case dbOpenDynaset
         RecordsetType = "dbOpenDynaset"
      Case dbOpenSnapshot
```

```
         RecordsetType = "dbOpenSnapshot"
      Case dbOpenForwardOnly
         RecordsetType = "dbOpenForwardOnly"
   End Select

End Function
```

This example demonstrates the **Type** property by returning the name of the constant corresponding 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.

```
Sub TypeX2()

   Dim dbsNorthwind As Database
   Dim fldLoop As Field

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   Debug.Print "Fields in Employees TableDef:"
   Debug.Print "    Type - Name"

   ' Enumerate Fields collection of Employees table.
   For Each fldLoop In _
      dbsNorthwind.TableDefs!Employees.Fields
      Debug.Print "        " & FieldType(fldLoop.Type) & _
         " - " & fldLoop.Name
   Next fldLoop

   dbsNorthwind.Close

End Sub

Function FieldType(intType As Integer) As String

   Select Case intType
      Case dbBoolean
         FieldType = "dbBoolean"
      Case dbByte
         FieldType = "dbByte"
      Case dbInteger
         FieldType = "dbInteger"
      Case dbLong
         FieldType = "dbLong"
      Case dbCurrency
         FieldType = "dbCurrency"
      Case dbSingle
         FieldType = "dbSingle"
      Case dbDouble
         FieldType = "dbDouble"
      Case dbDate
         FieldType = "dbDate"
      Case dbText
         FieldType = "dbText"
      Case dbLongBinary
         FieldType = "dbLongBinary"
      Case dbMemo
         FieldType = "dbMemo"
      Case dbGUID
```

```
        FieldType = "dbGUID"
   End Select

End Function
```

This example demonstrates the **Type** property by returning the name of the constant corresponding to the value of the **Type** property of all the **QueryDef** objects in Northwind. The QueryDefType function is required for this procedure to run.

```
Sub TypeX3()

   Dim dbsNorthwind As Database
   Dim qdfLoop As QueryDef

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   Debug.Print "QueryDefs in Northwind Database:"
   Debug.Print "    Type - Name"

   ' Enumerate QueryDefs collection of Northwind database.
   For Each qdfLoop In dbsNorthwind.QueryDefs
      Debug.Print "        " & _
         QueryDefType(qdfLoop.Type) & " - " & qdfLoop.Name
   Next qdfLoop

   dbsNorthwind.Close

End Sub

Function QueryDefType(intType As Integer) As String

   Select Case intType
      Case dbQSelect
         QueryDefType = "dbQSelect"
      Case dbQAction
         QueryDefType = "dbQAction"
      Case dbQCrosstab
         QueryDefType = "dbQCrosstab"
      Case dbQDelete
         QueryDefType = "dbQDelete"
      Case dbQUpdate
         QueryDefType = "dbQUpdate"
      Case dbQAppend
         QueryDefType = "dbQAppend"
      Case dbQMakeTable
         QueryDefType = "dbQMakeTable"
      Case dbQDDL
         QueryDefType = "dbQDDL"
      Case dbQSQLPassThrough
         QueryDefType = "dbQSQLPassThrough"
      Case dbQSetOperation
         QueryDefType = "dbQSetOperation"
      Case dbQSPTBulk
         QueryDefType = "dbQSPTBulk"
   End Select

End Function
```

Unique Property Example

This example sets the **Unique** property of a new **Index** object to **True**, and appends the **Index** to the **Indexes** collection of the Employees table. It then enumerates the **Indexes** collection of the **TableDef** and the **Properties** collection of each **Index**. The new **Index** will only allow one record with a particular combination of Country, LastName, and FirstName in the **TableDef**.

```
Sub UniqueX()

   Dim dbsNorthwind As Database
   Dim tdfEmployees As TableDef
   Dim idxNew As Index
   Dim idxLoop As Index
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set tdfEmployees = dbsNorthwind!Employees

   With tdfEmployees
      ' Create and append new Index object to the Indexes
      ' collection of the Employees table.
      Set idxNew = .CreateIndex("NewIndex")

      With idxNew
         .Fields.Append .CreateField("Country")
         .Fields.Append .CreateField("LastName")
         .Fields.Append .CreateField("FirstName")
         .Unique = True
      End With

      .Indexes.Append idxNew
      .Indexes.Refresh

      Debug.Print .Indexes.Count & " Indexes in " & _
         .Name & " TableDef"

      ' Enumerate Indexes collection of Employees table.
      For Each idxLoop In .Indexes
         Debug.Print "    " & idxLoop.Name

         ' Enumerate Properties collection of each Index
         ' object.
         For Each prpLoop In idxLoop.Properties
            Debug.Print "        " & prpLoop.Name & _
               " = " & IIf(prpLoop = "", "[empty]", prpLoop)
         Next prpLoop

      Next idxLoop

      ' Delete new Index because this is a demonstration.
      .Indexes.Delete idxNew.Name
   End With

   dbsNorthwind.Close

End Sub
```

Updatable Property Example

This example demonstrates the **Updatable** property for a **Database**, four types of **Recordset** objects, a **TableDef**, and a **QueryDef**.

```
Sub UpdatableX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      Debug.Print .Name
      Debug.Print "    Updatable = " & .Updatable

      ' Default is dbOpenTable.
      Set rstEmployees = .OpenRecordset("Employees")
      Debug.Print _
         "Table-type recordset from Employees table"
      Debug.Print "    Updatable = " & _
         rstEmployees.Updatable
      rstEmployees.Close

      Set rstEmployees = .OpenRecordset("Employees", _
         dbOpenDynaset)
      Debug.Print _
         "Dynaset-type recordset from Employees table"
      Debug.Print "    Updatable = " & _
         rstEmployees.Updatable
      rstEmployees.Close

      Set rstEmployees = .OpenRecordset("Employees", _
         dbOpenSnapshot)
      Debug.Print _
         "Snapshot-type recordset from Employees table"
      Debug.Print "    Updatable = " & _
         rstEmployees.Updatable
      rstEmployees.Close

      Set rstEmployees = .OpenRecordset("Employees", _
         dbOpenForwardOnly)
      Debug.Print _
         "Forward-only-type recordset from Employees table"
      Debug.Print "    Updatable = " & _
         rstEmployees.Updatable
      rstEmployees.Close

      Debug.Print "'" & .TableDefs(0).Name & "' TableDef"
      Debug.Print "    Updatable = " & _
         .TableDefs(0).Updatable

      Debug.Print "'" & .QueryDefs(0).Name & "' QueryDef"
      Debug.Print "    Updatable = " & _
         .QueryDefs(0).Updatable

      .Close
```

```
    End With

End Sub
```

V1xNullBehavior Property Example

This example converts a Microsoft Jet version 1.1 database file to a Microsoft Jet version 3.0 database file. During conversion, the **V1xNullBehavior** property is created and added to the **Properties** collection of the new database. The **Properties** collections of both database files are enumerated to show the change. Finally, the **V1xNullBehavior** property is deleted. This assumes that any applications will be modified to store **Null** values in empty Text and Memo fields rather than empty strings.

Note Unless you can obtain a Microsoft Jet version 1.1 file called "Nwind11.mdb," this procedure will not run.

```
Sub V1xNullBehaviorX()

   Dim dbsNorthwind As Database
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Nwind11.mdb")

   With dbsNorthwind
      Debug.Print .Name & ", version " & .Version
      ' Enumerate Properties collection of Northwind
      ' database.
      For Each prpLoop In .Properties
         On Error Resume Next
         If prpLoop <> "" Then Debug.Print "    " & _
            prpLoop.Name & " = " & prpLoop
         On Error GoTo 0
      Next prpLoop

      .Close
   End With

   DBEngine.CompactDatabase "Nwind11.mdb", _
      "Nwind30.mdb", , dbVersion30

   Set dbsNorthwind = OpenDatabase("Nwind30.mdb")

   With dbsNorthwind
      Debug.Print .Name & ", version " & .Version

      ' Enumerate Properties collection of compacted
      ' database. The V1xNullBehavior property cannot be
      ' referred to explicitly, that is,
      ' dbsNorthwind.V1xNullBehavior, but it can be accessed
      ' in loops or by string reference, that is,
      ' dbsNorthwind.Properties("V1xNullBehavior").
      For Each prpLoop In .Properties
         On Error Resume Next
         If prpLoop <> "" Then Debug.Print "    " & _
            prpLoop.Name & " = " & prpLoop
         On Error GoTo 0
      Next prpLoop

      .Properties.Delete "V1xNullBehavior"
      .Close
   End With
```

```
End Sub
```

ValidateOnSet Property Example

This example uses the **ValidateOnSet** property to demonstrate how one might trap for errors during data entry. The ValidateData function is required for this procedure to run.

```
Sub ValidateOnSetX()

   Dim dbsNorthwind As Database
   Dim fldDays As Field
   Dim rstEmployees As Recordset

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Create and append a new Field object to the Fields
   ' collection of the Employees table.
   Set fldDays = _
      dbsNorthwind.TableDefs!Employees.CreateField( _
      "DaysOfVacation", dbInteger, 2)
   fldDays.ValidationRule = "BETWEEN 1 AND 20"
   fldDays.ValidationText = _
      "Number must be between 1 and 20!"
   dbsNorthwind.TableDefs!Employees.Fields.Append fldDays

   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")

   With rstEmployees

      Do While True
         ' Add new record.
         .AddNew

         ' Get user input for three fields. Verify that the
         ' data do not violate the validation rules for any
         ' of the fields.
         If ValidateData(!FirstName, _
            "Enter first name.") = False Then Exit Do
         If ValidateData(!LastName, _
            "Enter last name.") = False Then Exit Do
         If ValidateData(!DaysOfVacation, _
            "Enter days of vacation.") = False Then Exit Do

         .Update
         .Bookmark = .LastModified
         Debug.Print !FirstName & " " & !LastName & _
            " - " & "DaysOfVacation = " & !DaysOfVacation

         ' Delete new record because this is a demonstration.
         .Delete
         Exit Do
      Loop

      ' Cancel AddNew method if any of the validation rules
      ' were broken.
      If .EditMode <> dbEditNone Then .CancelUpdate
      .Close
   End With
```

```
   ' Delete new field because this is a demonstration.
   dbsNorthwind.TableDefs!Employees.Fields.Delete _
      fldDays.Name
   dbsNorthwind.Close

End Sub

Function ValidateData(fldTemp As Field, _
   strMessage As String) As Boolean

   Dim strInput As String
   Dim errLoop As Error

   ValidateData = True
   ' ValidateOnSet is only read/write for Field objects in
   ' Recordset objects.
   fldTemp.ValidateOnSet = True

   Do While True
      strInput = InputBox(strMessage)
      If strInput = "" Then Exit Do
      ' Trap for errors when setting the Field value.
      On Error GoTo Err_Data
      If fldTemp.Type = dbInteger Then
         fldTemp = Val(strInput)
      Else
         fldTemp = strInput
      End If
      On Error GoTo 0
      If Not IsNull(fldTemp) Then Exit Do
   Loop

   If strInput = "" Then ValidateData = False

   Exit Function

Err_Data:

   If DBEngine.Errors.Count > 0 Then
      ' Enumerate the Errors collection. The description
      ' property of the last Error object will be set to
      ' the ValidationText property of the relevant
      ' field.
      For Each errLoop In DBEngine.Errors
         MsgBox "Error number: " & errLoop.Number & _
            vbCr & errLoop.Description
      Next errLoop
   End If

   Resume Next

End Function
```

ValidationRule and ValidationText Properties Example

This example creates a new **Field** object in the specified **TableDef** object and sets the **ValidationRule** and **ValidationText** properties based on the passed data. It also shows how the **ValidationRule** and **ValidationText** properties are used during actual data entry. The SetValidation function is required for this procedure to run.

```
Sub ValidationRuleX()

   Dim dbsNorthwind As Database
   Dim fldDays As Field
   Dim rstEmployees As Recordset
   Dim strMessage As String
   Dim strDays As String
   Dim errLoop As Error

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   ' Create a new field for the Employees TableDef object
   ' using the specified property settings.
   Set fldDays = _
      SetValidation(dbsNorthwind.TableDefs!Employees, _
      "DaysOfVacation", dbInteger, 2, "BETWEEN 1 AND 20", _
      "Number must be between 1 and 20!")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")

   With rstEmployees

      ' Enumerate Recordset. With each record, fill the new
      ' field with data supplied by the user.
      Do While Not .EOF
         .Edit
         strMessage = "Enter days of vacation for " & _
            !FirstName & " " & !LastName & vbCr & _
            "[" & !DaysOfVacation.ValidationRule & "]"

         Do While True
            ' Get user input.
            strDays = InputBox(strMessage)
            If strDays = "" Then
               .CancelUpdate
               Exit Do
            End If
            !DaysOfVacation = Val(strDays)

            ' Because ValidateOnSet defaults to False, the
            ' data in the buffer will be checked against the
            ' ValidationRule during Update.
            On Error GoTo Err_Rule
            .Update
            On Error GoTo 0

            ' If the Update method was successful, print the
            ' results of the data change.
            If .EditMode = dbEditNone Then
               Debug.Print !FirstName & " " & !LastName & _
                   " - " & "DaysOfVacation = " & _
```

```
                  !DaysOfVacation
               Exit Do
            End If

         Loop

         If strDays = "" Then Exit Do
         .MoveNext
      Loop

      .Close
   End With

   ' Delete new field because this is a demonstration.
   dbsNorthwind.TableDefs!Employees.Fields.Delete _
      fldDays.Name
   dbsNorthwind.Close

   Exit Sub

Err_Rule:

   If DBEngine.Errors.Count > 0 Then
      ' Enumerate the Errors collection.
      For Each errLoop In DBEngine.Errors
         MsgBox "Error number: " & _
            errLoop.Number & vbCr & _
            errLoop.Description
      Next errLoop
   End If

   Resume Next

End Sub

Function SetValidation(tdfTemp As TableDef, _
   strFieldName As String, intType As Integer, _
   intLength As Integer, strRule As String, _
   strText As String) As Field

   ' Create and append a new Field object to the Fields
   ' collection of the specified TableDef object.
   Set SetValidation = tdfTemp.CreateField(strFieldName, _
      intType, intLength)

   SetValidation.ValidationRule = strRule
   SetValidation.ValidationText = strText
   tdfTemp.Fields.Append SetValidation

End Function
```

Value Property Example

This example demonstrates the **Value** property with **Field** and **Property** objects.

```
Sub ValueX()

   Dim dbsNorthwind As Database
   Dim rstEmployees As Recordset
   Dim fldLoop As Field
   Dim prpLoop As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set rstEmployees = _
      dbsNorthwind.OpenRecordset("Employees")

   With rstEmployees
      Debug.Print "Field values in rstEmployees"
      ' Enumerate the Fields collection of the Employees
      ' table.
      For Each fldLoop In .Fields
         Debug.Print "    " & fldLoop.Name & " = ";
         Select Case fldLoop.Type
            Case dbLongBinary
               Debug.Print "[LongBinary]"
            Case dbMemo
               Debug.Print "[Memo]"
            Case Else
               ' Because Value is the default property of a
               ' Field object, the use of the actual keyword
               ' here is optional.
               Debug.Print fldLoop.Value
         End Select
      Next fldLoop

      Debug.Print "Property values in rstEmployees"
      ' Enumerate the Properties collection of the
      ' Recordset object.
      For Each prpLoop In .Properties
         On Error Resume Next
         ' Because Value is the default property of a
         ' Property object, the use of the actual keyword
         ' here is optional.
         If prpLoop <> "" Then Debug.Print "    " & _
            prpLoop.Name & " = " & prpLoop.Value
         On Error GoTo 0
      Next prpLoop

      .Close
   End With

   dbsNorthwind.Close

End Sub
```

AllPermissions, Permissions, and SystemDB Properties Example

This example uses the **SystemDB**, **AllPermissions**, and **Permissions** properties to show how users can have different levels of permissions depending on the permissions of the group to which they belong.

```
Sub AllPermissionsX()

   ' Ensure that the Microsoft Jet workgroup information
   ' file is available.
   DBEngine.SystemDB = "system.mdw"

   Dim dbsNorthwind As Database
   Dim ctrLoop As Container

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Enumerate Containers collection and display the current
   ' user and the permissions set for that user.
   For Each ctrLoop In dbsNorthwind.Containers
      With ctrLoop
         Debug.Print "Container: " & .Name
         Debug.Print "User: " & .UserName
         Debug.Print "    Permissions: " & .Permissions
         Debug.Print "    AllPermissions: " & _
            .AllPermissions
      End With
   Next ctrLoop

   dbsNorthwind.Close

End Sub
```

Attributes Property Example

This example displays the **Attributes** property for **Field**, **Relation**, and **TableDef** objects in the Northwind database.

```
Sub AttributesX()

   Dim dbsNorthwind As Database
   Dim fldLoop As Field
   Dim relLoop As Relation
   Dim tdfloop As TableDef

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind

      ' Display the attributes of a TableDef object's
      ' fields.
      Debug.Print "Attributes of fields in " & _
         .TableDefs(0).Name & " table:"
      For Each fldLoop In .TableDefs(0).Fields
         Debug.Print "     " & fldLoop.Name & " = " & _
            fldLoop.Attributes
      Next fldLoop

      ' Display the attributes of the Northwind database's
      ' relations.
      Debug.Print "Attributes of relations in " & _
         .Name & ":"
      For Each relLoop In .Relations
         Debug.Print "     " & relLoop.Name & " = " & _
            relLoop.Attributes
      Next relLoop

      ' Display the attributes of the Northwind database's
      ' tables.
      Debug.Print "Attributes of tables in " & .Name & ":"
      For Each tdfloop In .TableDefs
         Debug.Print "     " & tdfloop.Name & " = " & _
            tdfloop.Attributes
      Next tdfloop

      .Close
   End With

End Sub
```

BatchCollisionCount Property and Update Method Example

This example uses the **BatchCollisionCount** property and the **Update** method to demonstrate batch updating where any collisions are resolved by forcing the batch update.

```
Sub BatchX()

   Dim wrkMain As Workspace
   Dim conMain As Connection
   Dim rstTemp As Recordset
   Dim intLoop As Integer
   Dim strPrompt As String

   Set wrkMain = CreateWorkspace("ODBCWorkspace", _
      "admin", "", dbUseODBC)
   ' This DefaultCursorDriver setting is required for
   ' batch updating.
   wrkMain.DefaultCursorDriver = dbUseClientBatchCursor

   Set conMain = wrkMain.OpenConnection("Publishers", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")
   ' The following locking argument is required for
   ' batch updating.
   Set rstTemp = conMain.OpenRecordset( _
      "SELECT * FROM roysched", dbOpenDynaset, 0, _
      dbOptimisticBatch)

   With rstTemp
      ' Modify data in local recordset.
      Do While Not .EOF
         .Edit
         If !royalty <= 20 Then
            !royalty = !royalty - 4
         Else
            !royalty = !royalty + 2
         End If
         .Update
         .MoveNext
      Loop

      ' Attempt a batch update.
      .Update dbUpdateBatch

      ' If there are collisions, give the user the option
      ' of forcing the changes or resolving them
      ' individually.
      If .BatchCollisionCount > 0 Then
         strPrompt = "There are collisions. " & vbCr & _
            "Do you want the program to force " & _
            vbCr & "an update using the local data?"
         If MsgBox(strPrompt, vbYesNo) = vbYes Then _
            .Update dbUpdateBatch, True
      End If

      .Close
   End With
```

```
    conMain.Close
    wrkMain.Close

End Sub
```

BatchSize and UpdateOptions Properties Example

This example uses the **BatchSize** and **UpdateOptions** properties to control aspects of any batch updating for the specified **Recordset** object.

```
Sub BatchSizeX()

   Dim wrkMain As Workspace
   Dim conMain As Connection
   Dim rstTemp As Recordset

   Set wrkMain = CreateWorkspace("ODBCWorkspace", _
      "admin", "", dbUseODBC)
   ' This DefaultCursorDriver setting is required for
   ' batch updating.
   wrkMain.DefaultCursorDriver = dbUseClientBatchCursor

   Set conMain = wrkMain.OpenConnection("Publishers", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")
   ' The following locking argument is required for
   ' batch updating.
   Set rstTemp = conMain.OpenRecordset( _
      "SELECT * FROM roysched", dbOpenDynaset, 0, _
      dbOptimisticBatch)

   With rstTemp
      ' Increase the number of statements sent to the server
      ' during a single batch update, thereby reducing the
      ' number of times an update would have to access the
      ' server.
      .BatchSize = 25

      ' Change the UpdateOptions property so that the WHERE
      ' clause of any batched statements going to the server
      ' will include any updated columns in addition to the
      ' key column(s). Also, any modifications to records
      ' will be made by deleting the original record
      ' and adding a modified version rather than just
      ' modifying the original record.
      .UpdateOptions = dbCriteriaModValues + _
         dbCriteriaDeleteInsert

      ' Engage in batch updating using the new settings
      ' above.
      ' ...

      .Close
   End With

   conMain.Close
   wrkMain.Close

End Sub
```

CollatingOrder Property Example

This example displays the **CollatingOrder** property for the Northwind database and for individual fields in a table.

```
Sub CollatingOrderX()

   Dim dbsNorthwind As Database
   Dim fldLoop As Field

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      ' Show collating order of Northwind database.
      Debug.Print "Collating order of " & .Name & " = " & _
         .CollatingOrder

      ' Show collating order of a TableDef object's fields.
      Debug.Print "Collating order of fields in " & _
         .TableDefs(0).Name & " table:"
      For Each fldLoop In .TableDefs(0).Fields
         Debug.Print "    " & fldLoop.Name & " = " & _
            fldLoop.CollatingOrder
      Next fldLoop

      .Close
   End With

End Sub
```

ConflictTable Property Example

This example uses the **ConflictTable** property to report the table names that had conflicts during synchronization.

```
Sub ConflictTableX()

   Dim dbsNorthwind As Database
   Dim tdfTest As TableDef

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Enumerate TableDefs collection and check ConflictTable
   ' property of each.
   For Each tdfTest In dbsNorthwind.TableDefs
      If tdfTest.ConflictTable <> "" Then _
         Debug.Print tdfTest.Name & " had a conflict."
   Next tdfTest

   dbsNorthwind.Close

End Sub
```

This example opens a **Recordset** from the conflict table and one from the table that caused the conflict. It then processes the records in these tables, using the RequiredDate field to copy information from one table to the other depending on which record was more recently updated.

```
Sub ConflictTableX2(dbsResolve As Database)

   Dim tdfTest As TableDef
   Dim rstSource As Recordset
   Dim rstConflict As Recordset
   Dim fldLoop As Field

   Set tdfTest = dbsResolve.TableDefs("Orders")

   If tdfTest.ConflictTable <> "" Then

      Set rstSource = dbsResolve.OpenRecordset( _
         tdfTest.Name, dbOpenTable)
      Set rstConflict = dbsResolve.OpenRecordset( _
         tdfTest.ConflictTable, dbOpenTable)
      rstSource.Index = "[d_Guid]"
      rstConflict.MoveFirst

      Do Until rstConflict.EOF
         rstSource.Seek "=", rstConflict![s_Guid]
         If Not rstSource.NoMatch Then
            If rstSource!RequiredDate < _
                  rstConflict!RequiredDate Then
               On Error Resume Next
                  For Each fldLoop in rstConflict.Fields
                     fldLoop = rstSource(fldLoop.Name)
                  Next fldLoop
               On Error Goto 0
            End If
         End If
```

```
            rstConflict.Delete
            rstConflict.MoveNext
        Loop

        rstConflict.Close
        rstSource.Close
    End If

End Sub
```

Connect and SourceTableName Properties Example

This example uses the **Connect** and **SourceTableName** properties to link various external tables to a Microsoft Jet database. The ConnectOutput procedure is required for this procedure to run.

```
Sub ConnectX()

   Dim dbsTemp As Database
   Dim strMenu As String
   Dim strInput As String

   ' Open a Microsoft Jet database to which you will link
   ' a table.
   Set dbsTemp = OpenDatabase("DB1.mdb")

   ' Build menu text.
   strMenu = "Enter number for data source:" & vbCr
   strMenu = strMenu & _
      "   1. Microsoft Jet database" & vbCr
   strMenu = strMenu & _
      "   2. Microsoft FoxPro 3.0 table" & vbCr
   strMenu = strMenu & _
      "   3. dBASE table" & vbCr
   strMenu = strMenu & _
      "   4. Paradox table" & vbCr
   strMenu = strMenu & _
      "   M. (see choices 5-9)"

   ' Get user's choice.
   strInput = InputBox(strMenu)

   If UCase(strInput) = "M" Then

      ' Build menu text.
      strMenu = "Enter number for data source:" & vbCr
      strMenu = strMenu & _
         "   5. Microsoft Excel spreadsheet" & vbCr
      strMenu = strMenu & _
         "   6. Lotus spreadsheet" & vbCr
      strMenu = strMenu & _
         "   7. Comma-delimited text (CSV)" & vbCr
      strMenu = strMenu & _
         "   8. HTML table" & vbCr
      strMenu = strMenu & _
         "   9. Microsoft Exchange folder"

      ' Get user's choice.
      strInput = InputBox(strMenu)

   End If

   ' Call the ConnectOutput procedure. The third argument
   ' will be used as the Connect string, and the fourth
   ' argument will be used as the SourceTableName.
   Select Case Val(strInput)
      Case 1
         ConnectOutput dbsTemp, _
```

```
            "JetTable", _
            ";DATABASE=C:\My Documents\Northwind.mdb", _
            "Employees"
      Case 2
         ConnectOutput dbsTemp, _
            "FoxProTable", _
            "FoxPro 3.0;DATABASE=C:\FoxPro30\Samples", _
            "Q1Sales"
      Case 3
         ConnectOutput dbsTemp, _
            "dBASETable", _
            "dBase IV;DATABASE=C:\dBASE\Samples", _
            "Accounts"
      Case 4
         ConnectOutput dbsTemp, _
            "ParadoxTable", _
            "Paradox 3.X;DATABASE=C:\Paradox\Samples", _
            "Accounts"
      Case 5
         ConnectOutput dbsTemp, _
            "ExcelTable", _
            "Excel 5.0;" & _
               "DATABASE=C:\Excel\Samples\Q1Sales.xls", _
            "January Sales"
      Case 6
         ConnectOutput dbsTemp, _
            "LotusTable", _
            "Lotus WK3;" & _
               "DATABASE=C:\Lotus\Samples\Sales.xls", _
            "THIRDQTR"
      Case 7
         ConnectOutput dbsTemp, _
            "CSVTable", _
            "Text;DATABASE=C:\Samples", _
            "Sample.txt"
      Case 8
         ConnectOutput dbsTemp, _
            "HTMLTable", _
            "HTML Import;DATABASE=http://" & _
               "www.server1.com/samples/page1.html", _
            "Q1SalesData"
      Case 9
         ConnectOutput dbsTemp, _
            "ExchangeTable", _
            "Exchange 4.0;MAPILEVEL=" & _
               "Mailbox - Michelle Wortman (Exchange)" & _
               "|People\Important;", _
            "Jerry Wheeler"
   End Select

   dbsTemp.Close

End Sub

Sub ConnectOutput(dbsTemp As Database, _
   strTable As String, strConnect As String, _
   strSourceTable As String)
```

```
    Dim tdfLinked As TableDef
    Dim rstLinked As Recordset
    Dim intTemp As Integer

    ' Create a new TableDef, set its Connect and
    ' SourceTableName properties based on the passed
    ' arguments, and append it to the TableDefs collection.
    Set tdfLinked = dbsTemp.CreateTableDef(strTable)

    tdfLinked.Connect = strConnect
    tdfLinked.SourceTableName = strSourceTable
    dbsTemp.TableDefs.Append tdfLinked

    Set rstLinked = dbsTemp.OpenRecordset(strTable)

    Debug.Print "Data from linked table:"

    ' Display the first three records of the linked table.
    intTemp = 1
    With rstLinked
       Do While Not .EOF And intTemp <= 3
          Debug.Print , .Fields(0), .Fields(1)
          intTemp = intTemp + 1
          .MoveNext
       Loop
       If Not .EOF Then Debug.Print , "[additional records]"
       .Close
    End With

    ' Delete the linked table because this is a demonstration.
    dbsTemp.TableDefs.Delete strTable

End Sub
```

Container Property Example

This example displays the **Container** property for a variety of **Document** objects.

```
Sub ContainerPropertyX()

   Dim dbsNorthwind As Database
   Dim ctrLoop As Container

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Display the container name for the first Document
   ' object in each Container object's Documents collection.
   For Each ctrLoop In dbsNorthwind.Containers
      Debug.Print "Document: " & ctrLoop.Documents(0).Name
      Debug.Print "    Container = " & _
         ctrLoop.Documents(0).Container
   Next ctrLoop

   dbsNorthwind.Close

End Sub
```

Database Property Example

This example uses the **Database** property to show how code that used to access ODBC data through the Microsoft Jet database engine can be converted to use ODBCDirect **Connection** objects.

The OldDatabaseCode procedure uses a Microsoft Jet-connected data source to access an ODBC database.

```
Sub OldDatabaseCode()

   Dim wrkMain As Workspace
   Dim dbsPubs As Database
   Dim prpLoop As Property

   ' Create Microsoft Jet Workspace object.
   Set wrkMain = CreateWorkspace("", "admin", "", dbUseJet)

   ' Open a Database object based on information in
   ' the connect string.
   Set dbsPubs = wrkMain.OpenDatabase("Publishers", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")

   ' Enumerate the Properties collection of the Database
   ' object.
   With dbsPubs
      Debug.Print "Database properties for " & _
         .Name & ":"

      On Error Resume Next
      For Each prpLoop In .Properties
         If prpLoop.Name = "Connection" Then
            ' Property actually returns a Connection object.
            Debug.Print "    Connection[.Name] = " & _
               .Connection.Name
         Else
            Debug.Print "    " & prpLoop.Name & " = " & _
               prpLoop
         End If
      Next prpLoop
      On Error GoTo 0

   End With

   dbsPubs.Close
   wrkMain.Close

End Sub
```

The NewDatabaseCode example opens a **Connection** object in an ODBCDirect workspace. It then assigns the **Database** property of the **Connection** object to an object variable with the same name as the data source in the old procedure. None of the subsequent code has to be changed as long as it doesn't use any features specific to Microsoft Jet workspaces.

```
Sub NewDatabaseCode()

   Dim wrkMain As Workspace
   Dim conPubs As Connection
```

```
    Dim dbsPubs As Database
    Dim prpLoop As Property

    ' Create ODBCDirect Workspace object instead of Microsoft
    ' Jet Workspace object.
    Set wrkMain = CreateWorkspace("", "admin", "", dbUseODBC)

    ' Open Connection object based on information in
    ' the connect string.
    Set conPubs = wrkMain.OpenConnection("Publishers", _
       dbDriverNoPrompt, False, _
       "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")
    ' Assign the Database property to the same object
    ' variable as in the old code.
    Set dbsPubs = conPubs.Database

    ' Enumerate the Properties collection of the Database
    ' object. From this point on, the code is the same as the
    ' old example.
    With dbsPubs
       Debug.Print "Database properties for " & _
          .Name & ":"

       On Error Resume Next
       For Each prpLoop In .Properties
          If prpLoop.Name = "Connection" Then
             ' Property actually returns a Connection object.
             Debug.Print "    Connection[.Name] = " & _
                .Connection.Name
          Else
             Debug.Print "    " & prpLoop.Name & " = " & _
                prpLoop
          End If
       Next prpLoop
       On Error GoTo 0

    End With

    dbsPubs.Close
    wrkMain.Close

End Sub
```

DefaultType Property Example

This example uses the **DefaultType** property to predetermine what type of **Workspace** object will be created when you call the **CreateWorkspace** method. The TypeOutput function is required for this procedure to run.

```
Sub DefaultTypeX()

   Dim wrkODBC As Workspace
   Dim wrkJet As Workspace
   Dim prpLoop As Property

   ' Set DefaultType property and create Workspace object
   ' without specifying a type.
   DBEngine.DefaultType = dbUseODBC
   Set wrkODBC = CreateWorkspace("ODBCWorkspace", _
      "admin", "")

   Debug.Print "DBEngine.DefaultType = " & _
      TypeOutput(DBEngine.DefaultType)
   With wrkODBC
      ' Enumerate Properties collection of Workspace object.
      Debug.Print "Properties of " & .Name
      On Error Resume Next
      For Each prpLoop In .Properties
         Debug.Print "    " & prpLoop.Name & " = " & prpLoop
         If prpLoop.Name = "Type" Then Debug.Print _
            "         (" & TypeOutput(prpLoop.Value) & ")"
      Next prpLoop
      On Error GoTo 0
   End With

   ' Set DefaultType property and create Workspace object
   ' without specifying a type.
   DBEngine.DefaultType = dbUseJet
   Set wrkJet = CreateWorkspace("JetWorkspace", "admin", "")

   Debug.Print "DBEngine.DefaultType = " & _
      TypeOutput(DBEngine.DefaultType)
   With wrkJet
      ' Enumerate Properties collection of Workspace object.
      Debug.Print "Properties of " & .Name
      On Error Resume Next
      For Each prpLoop In .Properties
         Debug.Print "    " & prpLoop.Name & " = " & prpLoop
         If prpLoop.Name = "Type" Then Debug.Print _
            "         (" & TypeOutput(prpLoop.Value) & ")"
      Next prpLoop
      On Error GoTo 0
   End With

   wrkODBC.Close
   wrkJet.Close

End Sub

Function TypeOutput(intTemp As Integer) As String
```

```
    If intTemp = dbUseJet Then
        TypeOutput = "dbUseJet"
    Else
        TypeOutput = "dbUseODBC"
    End If

End Function
```

DefaultUser, DefaultPassword Properties Example

This example sets the **DefaultUser** and **DefaultPassword** properties which will determine the settings for the default **Workspace** object.

```
Sub DefaultUserX()

   ' Set the DefaultUser and DefaultPassword properties for
   ' the DBEngine object.
   DBEngine.DefaultUser = "NewUser"
   DBEngine.DefaultPassword = ""

   Debug.Print _
      "Setting DBEngine.DefaultUser to 'NewUser'..."
   Debug.Print _
      "Setting DBEngine.DefaultPassword to " & _
         "[zero-length string]..."

   Dim wrkJet As Workspace
   Dim wrkLoop As Workspace
   Dim prpLoop As Property

   Set wrkJet = CreateWorkspace("JetWorkspace", "admin", _
      "", dbUseJet)

   ' Enumerate Workspaces collection.
   On Error Resume Next
   For Each wrkLoop In Workspaces
      Debug.Print "Workspace: " & wrkLoop.Name
      ' Enumerate Properties collection of each Workspace
      ' object.
      For Each prpLoop In wrkLoop.Properties
         Debug.Print "    " & prpLoop.Name & " = " & prpLoop
      Next prpLoop
   Next wrkLoop
   On Error GoTo 0

   wrkJet.Close

End Sub
```

DesignMasterID Property Example

This example sets the **DesignMasterID** property to the **ReplicaID** property setting of another database, making that database the Design Master in the replica set. The old and new Design Masters are synchronized to update the design change. For this code to work, you must create a Design Master and replica, include their names and paths as appropriate, and run this code from a database other than the old or new Design Master.

```
Sub SetNewDesignMaster(strOldDM as String, _
   strNewDM as String)

   Dim dbsOld As Database
   Dim dbsNew As Database

   ' Open the current Design Master in exclusive mode.
   Set dbsOld = OpenDatabase(strOldDM, True)

   ' Open the database that will become the new
   ' Design Master.
   Set dbsNew = OpenDatabase(strNewDM)

   ' Make the new database the Design Master.
   dbsOld.DesignMasterID = dbsNew.ReplicaID

   ' Synchronize the old Design Master with the new
   ' Design Master, and allow two-way exchanges.
   dbsOld.Synchronize strNewDM, dbRepImpExpChanges
   dbsOld.Close
   dbsNew.Close

End Sub
```

Direction Property Example

This example uses the **Direction** property to configure the parameters of a query to an ODBC data source.

```
Sub DirectionX()

   Dim wrkMain As Workspace
   Dim conMain As Connection
   Dim qdfTemp As QueryDef
   Dim rstTemp As Recordset
   Dim strSQL As String
   Dim intLoop As Integer

   ' Create ODBC workspace and open a connection to a
   ' Microsoft SQL Server database.
   Set wrkMain = CreateWorkspace("ODBCWorkspace", _
      "admin", "", dbUseODBC)
   Set conMain = wrkMain.OpenConnection("Publishers", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")

   ' Set SQL string to call the stored procedure
   ' getempsperjob.
   strSQL = "{ call getempsperjob (?, ?) }"

   Set qdfTemp = conMain.CreateQueryDef("", strSQL)

   With qdfTemp
      ' Indicate that the two query parameters will only
      ' pass information to the stored procedure.
      .Parameters(0).Direction = dbParamInput
      .Parameters(1).Direction = dbParamInput

      ' Assign initial parameter values.
      .Parameters(0) = "0877"
      .Parameters(1) = 0

      Set rstTemp = .OpenRecordset()

      With rstTemp
         ' Loop through all valid values for the second
         ' parameter. For each value, requery the recordset
         ' to obtain the correct results and then print out
         ' the contents of the recordset.
         For intLoop = 1 To 14
            qdfTemp.Parameters(1) = intLoop
            .Requery
            Debug.Print "Publisher = " & _
               qdfTemp.Parameters(0) & _
               ", job = " & intLoop
            Do While Not .EOF
               Debug.Print , .Fields(0), .Fields(1)
               .MoveNext
            Loop
         Next intLoop
         .Close
```

```
        End With

    End With

    conMain.Close
    wrkMain.Close

End Sub
```

Error Object, Errors Collection, and Description, Number, Source, HelpFile, and HelpContext Properties Example

This example forces an error, traps it, and displays the **Description**, **Number**, **Source**, **HelpContext**, and **HelpFile** properties of the resulting **Error** object.

```
Sub DescriptionX()

   Dim dbsTest As Database

   On Error GoTo ErrorHandler

   ' Intentionally trigger an error.
   Set dbsTest = OpenDatabase("NoDatabase")

   Exit Sub

ErrorHandler:
   Dim strError As String
   Dim errLoop As Error

   ' Enumerate Errors collection and display properties of
   ' each Error object.
   For Each errLoop In Errors
      With errLoop
         strError = _
            "Error #" & .Number & vbCr
         strError = strError & _
            "     " & .Description & vbCr
         strError = strError & _
            "     (Source: " & .Source & ")" & vbCr
         strError = strError & _
            "Press F1 to see topic " & .HelpContext & vbCr
         strError = strError & _
            "     in the file " & .HelpFile & "."
      End With
      MsgBox strError
   Next

   Resume Next

End Sub
```

Inherit Property Example

This example sets the Tables container's **Inherit** property to True so that any subsequently created **Document** objects in the Tables container will have the same security settings as the Tables container.

```
Sub InheritX()

   Dim dbsNorthwind As Database
   Dim conTables As Container

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set conTables = dbsNorthwind.Containers("Tables")

   ' By setting the Inherit property of the Tables container
   ' to true and setting its permissions, any new Document
   ' object in this container will inherit the same
   ' permissions setting.
   conTables.Inherit = True
   conTables.Permissions = dbSecWriteSec

   dbsNorthwind.Close

End Sub
```

Inherited Property Example

This example use the **Inherited** property to determine if a user-defined **Property** object was created for a **Recordset** object or for some underlying object.

```
Sub InheritedX()

   Dim dbsNorthwind As Database
   Dim tdfTest As TableDef
   Dim rstTest As Recordset
   Dim prpNew As Property
   Dim prpLoop As Property

   ' Create a new property for a saved TableDef object, then
   ' open a recordset from that TableDef object.
   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set tdfTest = dbsNorthwind.TableDefs(0)
   Set prpNew = tdfTest.CreateProperty("NewProperty", _
      dbBoolean, True)
   tdfTest.Properties.Append prpNew
   Set rstTest = tdfTest.OpenRecordset(dbOpenForwardOnly)

   ' Show Name and Inherited property of the new Property
   ' object in the TableDef.
   Debug.Print "NewProperty of " & tdfTest.Name & _
      " TableDef:"
   Debug.Print "    Inherited = " & _
      tdfTest.Properties("NewProperty").Inherited

   ' Show Name and Inherited property of the new Property
   ' object in the Recordset.
   Debug.Print "NewProperty of " & rstTest.Name & _
      " Recordset:"
   Debug.Print "    Inherited = " & _
      rstTest.Properties("NewProperty").Inherited

   ' Delete new TableDef because this is a demonstration.
   tdfTest.Properties.Delete prpNew.Name
   dbsNorthwind.Close

End Sub
```

IniPath Property Example

This example sets the path in the **IniPath** property to an application's key in the Windows Registry.

```
Sub IniPathX()

   ' Change the IniPath property to point to a different
   ' section of the Windows Registry for settings
   ' information.
   Debug.Print "Original IniPath setting = " & _
      IIf(DBEngine.IniPath = "", "[Empty]", _
      DBEngine.IniPath)
   DBEngine.IniPath = _
      "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\" & _
      "Jet\3.5\ISAM Formats\FoxPro 3.0"
   Debug.Print "New IniPath setting = " & _
      IIf(DBEngine.IniPath = "", "[Empty]", _
      DBEngine.IniPath)

End Sub
```

IsolateODBCTrans Property Example

This example opens three ODBCDirect workspaces and sets their **IsolateODBCTrans** properties to True so that multiple transactions to the same data source will be isolated from each other.

```
Sub IsolateODBCTransX()

   DBEngine.DefaultType = dbUseJet

   Dim wrkJet1 As Workspace
   Dim wrkJet2 As Workspace
   Dim wrkJet3 As Workspace

   ' Open three ODBCDirect workspaces to separate
   ' transactions involving the same ODBC data source.
   Set wrkJet1 = CreateWorkspace("", "admin", "")
   wrkJet1.IsolateODBCTrans = True

   Set wrkJet2 = CreateWorkspace("", "admin", "")
   wrkJet2.IsolateODBCTrans = True

   Set wrkJet3 = CreateWorkspace("", "admin", "")
   wrkJet3.IsolateODBCTrans = True

   wrkJet1.Close
   wrkJet2.Close
   wrkJet3.Close

End Sub
```

KeepLocal Property Example

The following example appends the **KeepLocal** property to the properties collection of a document object for the Utilities module in the Northwind database. You set this property on an object (such as a table) before a database is made replicable. When the database is converted to a Design Master, the object you specified to remain local will not be dispersed to other members of the replica set. Adjust the path to Northwind.mdb as appropriate to its location on your computer.

```
Sub KeepLocalNWObjectX()

   Dim dbsNorthwind As Database
   Dim docTemp As Document
   Dim prpTemp As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   Set docTemp = dbsNorthwind.Containers("Modules"). _
      Documents("Utility Functions")
   Set prpTemp = doc.CreateProperty("KeepLocal", _
      dbText, "T")
   docTemp.Properties.Append prpTemp
   dbsNorthwind.Close

End Sub
```

The following code sets the **KeepLocal** property on the specified **TableDef** object to `"T"`. If the **KeepLocal** property doesn't exist, it is created and appended to the table's **Properties** collection, and given a value of `"T"`.

```
Sub SetKeepLocal(tdfTemp As TableDef)

   On Error GoTo ErrHandler

   tdfTemp.Properties("KeepLocal") = "T"

   On Error GoTo 0

   Exit Sub

ErrHandler:

   Dim prpNew As Property

   If Err.Number = 3270 Then
      Set prpNew = tdfTemp.CreateProperty("KeepLocal", _
         dbText, "T")
      tdfTemp.Properties.Append prpNew
   Else
      MsgBox "Error " & Err & ": " & Error
   End If

End Sub
```

LoginTimeout Property Example

This example sets the **LoginTimeout** property of the **DBEngine** object to 120 seconds. It then opens three ODBCDirect workspaces and modifies their **LoginTimeout** properties from the default inherited from the **DBEngine** object.

```
Sub LoginTimeoutX()

   ' Change the default LoginTimeout value.
   DBEngine.LoginTimeout = 120

   Dim wrkODBC1 As Workspace
   Dim wrkODBC2 As Workspace
   Dim wrkODBC3 As Workspace

   Set wrkODBC1 = CreateWorkspace("", "admin", "", _
      dbUseODBC)
   Set wrkODBC2 = CreateWorkspace("", "admin", "", _
      dbUseODBC)
   Set wrkODBC3 = CreateWorkspace("", "admin", "", _
      dbUseODBC)

   ' Change the LoginTimeout of the individual ODBCDirect
   ' workspaces for 60 seconds, the default time (120
   ' seconds), and no timeout.
   wrkODBC1.LoginTimeout = 60
   wrkODBC2.LoginTimeout = -1
   wrkODBC2.LoginTimeout = 0

   wrkODBC1.Close
   wrkODBC2.Close
   wrkODBC3.Close

End Sub
```

LogMessages and ReturnsRecords Properties Example

This example uses the **LogMessages** and **ReturnsRecords** properties to create a pass-through query that will return data and any messages generated by the remote server.

```
Sub LogMessagesX()

   Dim wrkJet As Workspace
   Dim dbsCurrent As Database
   Dim qdfTemp As QueryDef
   Dim prpNew As Property
   Dim rstTemp As Recordset

   ' Create Microsoft Jet Workspace object.
   Set wrkJet = CreateWorkspace("", "admin", "", dbUseJet)

   Set dbsCurrent = wrkJet.OpenDatabase("DB1.mdb")

   ' Create a QueryDef that will log any messages from the
   ' server in temporary tables.
   Set qdfTemp = dbsCurrent.CreateQueryDef("NewQueryDef")
   qdfTemp.Connect = _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers"
   qdfTemp.SQL = "SELECT * FROM stores"
   qdfTemp.ReturnsRecords = True
   Set prpNew = qdfTemp.CreateProperty("LogMessages", _
      dbBoolean, True)
   qdfTemp.Properties.Append prpNew

   ' Execute query and display results.
   Set rstTemp = qdfTemp.OpenRecordset()

   Debug.Print "Contents of recordset:"
   With rstTemp
      Do While Not .EOF
         Debug.Print , .Fields(0), .Fields(1)
         .MoveNext
      Loop
      .Close
   End With

   ' Delete new QueryDef because this is a demonstration.
   dbsCurrent.QueryDefs.Delete qdfTemp.Name
   dbsCurrent.Close
   wrkJet.Close

End Sub
```

MaxRecords Property Example

This example uses the **MaxRecords** property to set a limit on how many records are returned by a query on an ODBC data source.

```
Sub MaxRecordsX()

   Dim dbsCurrent As Database
   Dim qdfPassThrough As QueryDef
   Dim qdfLocal As QueryDef
   Dim rstTemp As Recordset

   ' Open a database from which QueryDef objects can be
   ' created.
   Set dbsCurrent = OpenDatabase("DB1.mdb")

   ' Create a pass-through query to retrieve data from
   ' a Microsoft SQL Server database.
   Set qdfPassThrough = _
      dbsCurrent.CreateQueryDef("")

   ' Set the properties of the new query, limiting the
   ' number of returnable records to 20.
   qdfPassThrough.Connect = _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers"
   qdfPassThrough.SQL = "SELECT * FROM titles"
   qdfPassThrough.ReturnsRecords = True
   qdfPassThrough.MaxRecords = 20

   Set rstTemp = qdfPassThrough.OpenRecordset()

   ' Display results of query.
   Debug.Print "Query results:"
   With rstTemp
      Do While Not .EOF
         Debug.Print , .Fields(0), .Fields(1)
         .MoveNext
      Loop
      .Close
   End With

   dbsCurrent.Close

End Sub
```

Name Property Example

This example uses the **Name** property to give a name to a newly created object, to show what objects are in a given collection, and to delete an object from a collection.

```
Sub NameX()

   Dim dbsNorthwind As Database
   Dim qdfNew As QueryDef
   Dim qdfLoop As QueryDef

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      ' Create a new permanent QueryDef object and append it
      ' to the QueryDefs collection.
      Set qdfNew = .CreateQueryDef()
      qdfNew.Name = "NewQueryDef"
      qdfNew.SQL = "SELECT * FROM Employees"
      .QueryDefs.Append qdfNew

      ' Enumerate the QueryDefs collection to display the
      ' names of the QueryDef objects.
      Debug.Print "Names of queries in " & .Name

      For Each qdfLoop In .QueryDefs
         Debug.Print "    " & qdfLoop.Name
      Next qdfLoop

      ' Delete new QueryDef object because this is a
      ' demonstration.
      .QueryDefs.Delete qdfNew.Name

      .Close
   End With

End Sub
```

ODBCTimeout and QueryTimeout Properties Example

This example uses the **ODBCTimeout** and **QueryTimeout** properties to show how the **QueryTimeout** setting on a **Database** object sets the default **ODBCTimeout** setting on any **QueryDef** objects created from the **Database** object.

```
Sub ODBCTimeoutX()

   Dim dbsCurrent As Database
   Dim qdfStores As QueryDef
   Dim rstStores As Recordset

   Set dbsCurrent = OpenDatabase("Northwind.mdb")

   ' Change the default QueryTimeout of the Northwind
   ' database.
   Debug.Print "Default QueryTimeout of Database: " & _
      dbsCurrent.QueryTimeout
   dbsCurrent.QueryTimeout = 30
   Debug.Print "New QueryTimeout of Database: " & _
      dbsCurrent.QueryTimeout

   ' Create a new QueryDef object.
   Set qdfStores = dbsCurrent.CreateQueryDef("Stores", _
      "SELECT * FROM stores")
   qdfStores.Connect = _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers"

   ' Change the ODBCTimeout setting of the new QueryDef
   ' object from its default setting.
   Debug.Print "Default ODBCTimeout of QueryDef: " & _
      qdfStores.ODBCTimeout
   qdfStores.ODBCTimeout = 0
   Debug.Print "New ODBCTimeout of QueryDef: " & _
      qdfStores.ODBCTimeout

   ' Execute the query and display the results.
   Set rstStores = qdfStores.OpenRecordset()

   Debug.Print "Contents of recordset:"
   With rstStores
      Do While Not .EOF
         Debug.Print , .Fields(0), .Fields(1)
         .MoveNext
      Loop
      .Close
   End With

   ' Delete new QueryDef because this is a demonstration.
   dbsCurrent.QueryDefs.Delete qdfStores.Name
   dbsCurrent.Close

End Sub
```

Owner and SystemDB Properties Example

This example uses the **Owner** and **SystemDB** properties to show the owners of a variety of **Document** objects.

```
Sub OwnerX()

   ' Ensure that the Microsoft Jet workgroup file is
   ' available.
   DBEngine.SystemDB = "system.mdw"

   Dim dbsNorthwind As Database
   Dim ctrLoop As Container

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      Debug.Print "Document owners:"
      ' Enumerate Containers collection and show the owner
      ' of the first Document in each container's Documents
      ' collection.
      For Each ctrLoop In .Containers
         With ctrLoop
            Debug.Print "    [" & .Documents(0).Name & _
               "] in [" & .Name & _
               "] container owned by [" & _
               .Documents(0).Owner & "]"
         End With
      Next ctrLoop

      .Close
   End With

End Sub
```

PartialReplica Property Example

The following code example uses the **PartialReplica** property to replicate all records representing orders from customers in California:

```
Sub PartialReplicaX()

   ' Assumptions: dbsTemp is the partial replica and
   ' appropriate relationships already exist between
   ' the tables.

   Dim tdfOrders As TableDef
   Dim relCustOrd As Relation
   Dim dbsTemp As Database
   Dim relLoop As Relation

   Set dbsTemp = OpenDatabase("Northwind.mdb")
   Set tdfOrders = dbsTemp.TableDefs("Orders")

   ' Find the "Customers to Orders" Relation object.
   For Each relLoop In dbsTemp.Relations
      If relLoop.Table = "Customers" And _
            relLoop.ForeignTable = "Orders" Then
         ' Set the Relation object's PartialReplica
         ' property to True.
         relLoop.PartialReplica = True
         Exit For
      End If
   Next relLoop

End Sub
```

Note If you have set a replica filter and a replica relation on the same table, the two act in combination as a logical OR operation, not a logical AND operation. For instance, in the preceding example, the records exchanged during synchronization are all orders greater than $1000 OR all orders from the California region, not all orders from the California region that are over $1000.

Prepare Property Example

This example uses the **Prepare** property to specify that a query should be executed directly rather than first creating a temporary stored procedure on the server.

```
Sub PrepareX()

   Dim wrkODBC As Workspace
   Dim conPubs As Connection
   Dim qdfTemp As QueryDef
   Dim rstTemp As Recordset

   ' Create ODBCDirect Workspace object and open Connection
   ' object.
   Set wrkODBC = CreateWorkspace("", _
      "admin", "", dbUseODBC)
   Set conPubs = wrkODBC.OpenConnection("Publishers", , , _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")

   Set qdfTemp = conPubs.CreateQueryDef("")

   With qdfTemp
      ' Because you will only run this query once, specify
      ' the ODBC SQLExecDirect API function. If you do
      ' not set this property before you set the SQL
      ' property, the ODBC SQLPrepare API function will
      ' be called anyway which will nullify any
      ' performance gain.
      .Prepare = dbQUnprepare
      .SQL = "UPDATE roysched " & _
         "SET royalty = royalty * 2 " & _
         "WHERE title_id LIKE 'BU____' OR " & _
         "title_id LIKE 'PC____'"
      .Execute
   End With

   Debug.Print "Query results:"

   ' Open recordset containing modified records.
   Set rstTemp = conPubs.OpenRecordset( _
      "SELECT * FROM roysched " & _
      "WHERE title_id LIKE 'BU____' OR " & _
      "title_id LIKE 'PC____'")

   ' Enumerate recordset.
   With rstTemp
      Do While Not .EOF
         Debug.Print , !title_id, !lorange, _
            !hirange, !royalty
         .MoveNext
      Loop
      .Close
   End With

   conPubs.Close
   wrkODBC.Close
```

```
End Sub
```

RecordStatus and DefaultCursorDriver Properties Example

This example uses the **RecordStatus** and **DefaultCursorDriver** properties to show how changes to a local **Recordset** are tracked during batch updating. The RecordStatusOutput function is required for this procedure to run.

```
Sub RecordStatusX()

   Dim wrkMain As Workspace
   Dim conMain As Connection
   Dim rstTemp As Recordset

   Set wrkMain = CreateWorkspace("ODBCWorkspace", _
      "admin", "", dbUseODBC)
   ' This DefaultCursorDriver setting is required for
   ' batch updating.
   wrkMain.DefaultCursorDriver = dbUseClientBatchCursor

   Set conMain = wrkMain.OpenConnection("Publishers", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")
   ' The following locking argument is required for
   ' batch updating.
   Set rstTemp = conMain.OpenRecordset( _
      "SELECT * FROM authors", dbOpenDynaset, 0, _
      dbOptimisticBatch)

   With rstTemp
      .MoveFirst
      Debug.Print "Original record: " & !au_lname
      Debug.Print , RecordStatusOutput2(.RecordStatus)

      .Edit
      !au_lname = "Bowen"
      .Update
      Debug.Print "Edited record: " & !au_lname
      Debug.Print , RecordStatusOutput2(.RecordStatus)

      .AddNew
      !au_lname = "NewName"
      .Update
      Debug.Print "New record: " & !au_lname
      Debug.Print , RecordStatusOutput2(.RecordStatus)

      .Delete
      Debug.Print "Deleted record: " & !au_lname
      Debug.Print , RecordStatusOutput2(.RecordStatus)

      ' Close the local recordset without updating the
      ' data on the server.
      .Close
   End With

   conMain.Close
   wrkMain.Close

End Sub
```

```
Function RecordStatusOutput(lngTemp As Long) As String

   Dim strTemp As String

   strTemp = ""

   ' Construct an output string based on the RecordStatus
   ' value.
If lngTemp = dbRecordUnmodified Then _
      strTemp = "[dbRecordUnmodified]"
   If lngTemp = dbRecordModified Then _
      strTemp = "[dbRecordModified]"
   If lngTemp = dbRecordNew Then _
      strTemp = "[dbRecordNew]"
   If lngTemp = dbRecordDeleted Then _
      strTemp = "[dbRecordDeleted]"
   If lngTemp = dbRecordDBDeleted Then _
      strTemp = "[dbRecordDBDeleted]"

   RecordStatusOutput = strTemp

End Function
```

Replicable Property Example

These examples illustrate two situations for using the **Replicable** Property — on a database and on an object in the database.

This example makes Northwind.mdb a replicable database. It's recommended that you make a back-up copy of Northwind before running this code, and that you adjust the path to Northwind as appropriate to its location on your computer.

```
Sub MakeDesignMasterX()

   Dim dbsNorthwind As Database
   Dim prpNew As Property

   ' Open database for exclusive access.
   Set dbsNorthwind = OpenDatabase("Northwind.mdb", _
      True)

   With dbsNorthwind

      ' If Replicable property doesn't exist, create it.
      ' Turn off error handling in case property exists.
      On Error Resume Next
      Set prpNew = .CreateProperty("Replicable", _
         dbText, "T")
      .Properties.Append prpNew

      ' Set database Replicable property to True.
      .Properties("Replicable") = "T"

      .Close

   End With

End Sub
```

This example creates a new **TableDef** and then makes it replicable. The database must first be replicable for this procedure to work.

```
Sub CreateReplLocalTableX()

   Dim dbsNorthwind As Database
   Dim tdfNew As TableDef
   Dim fldNew As Field
   Dim prpNew As Property

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")
   ' Create a new TableDef named "Taxes".
   Set tdfNew = dbsNorthwind.CreateTableDef("Taxes")
   ' Define a text field named "Grade".
   Set fldNew = tdfNew.CreateField("Grade", dbText, 3)
   ' Append new field to the TableDef.
   tdfNew.Fields.Append fldNew

   ' Add the new TableDef to the database.
   dbsNorthwind.TableDefs.Append tdfNew

   ' Create a new Replicable property for new TableDef.
```

```
   Set prpNew = tdfNew.CreateProperty("Replicable", _
      dbText, "T")
   ' Append the Replicable property to the new
   ' TableDef.
   tdfNew.Properties.Append prpNew
   dbsNorthwind.Close

End Sub
```

The following code sets the **Replicable** property on the specified **TableDef** object to `"T"`. If the property does not exist, it is created and appended to the table's **Properties** collection, and given a value of `"T"`.

```
Sub SetReplicable(tdfTemp As TableDef)

   On Error GoTo ErrHandler

   tdfTemp.Properties("Replicable") = "T"

   On Error GoTo 0

   Exit Sub

ErrHandler:

   Dim prpNew As Property

   If Err.Number = 3270 Then
      Set prpNew = tdfTemp.CreateProperty("Replicable", _
         dbText, "T")
      tdfTemp.Properties.Append prpNew
   Else
      MsgBox "Error " & Err & ": " & Error
   End If

End Sub
```

ReplicaFilter Property Example

The following example uses the **ReplicaFilter** property to replicate only customer records from the California region.

```
Sub ReplicaFilterX()

   ' This example assumes the current open database
   ' is the replica.
   Dim tdfCustomers As TableDef
   Dim strFilter As String
   Dim dbsTemp As Database

   Set dbsTemp = OpenDatabase("Northwind.mdb")
   Set tdfCustomers = dbsTemp.TableDefs("Customers")

   ' Synchronize with full replica
   ' before setting replica filter.
   dbsTemp.Synchronize "C:\SALES\FY96.MDB"

   strFilter = "Region = 'CA'"
   tdfCustomers.ReplicaFilter = strFilter
   dbsTemp.PopulatePartial "C:\SALES\FY96.MDB"

   ' Now remove the replica filter (for example purposes
   ' only).
   tdfCustomers.ReplicaFilter = False
   ' Repopulate the database.
   dbsTemp.PopulatePartial "C:\SALES\DATA96.MDB"

End Sub
```

ReplicaID Property Example

This example makes a replica from the Design Master of Northwind.mdb, and then returns the replica's **ReplicaID**, which is automatically created by the Microsoft Jet database engine. (If you have not yet created a Design Master of Northwind, refer to the **Replicable** property, or change the name of the database in the code to an existing Design Master.)

```
Sub MakeReplicaReplicaIDX()

   Dim dbsNorthwind As Database
   Dim prpReplicaID As Property
   Dim dbsReplica As Database

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   ' Makes a new replica.
   dbsNorthwind.MakeReplica "Nwreplica2.mdb", _
      "second replica"
   dbsNorthwind.Close

   ' Opens the new replica to read its ReplicaID.
   Set dbsReplica = OpenDatabase("Nwreplica2.mdb")

   Debug.Print dbsReplica.ReplicaID
   dbsReplica.Close

End Sub
```

Required Property Example

This example uses the **Required** property to report which fields in three different tables must contain data in order for a new record to be added. The RequiredOutput procedure is required for this procedure to run.

```
Sub RequiredX()

   Dim dbsNorthwind As Database
   Dim tdfloop As TableDef

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   With dbsNorthwind
      ' Show which fields are required in the Fields
      ' collections of three different TableDef objects.
      RequiredOutput .TableDefs("Categories")
      RequiredOutput .TableDefs("Customers")
      RequiredOutput .TableDefs("Employees")
      .Close
   End With

End Sub

Sub RequiredOutput(tdfTemp As TableDef)

   Dim fldLoop As Field

   ' Enumerate Fields collection of the specified TableDef
   ' and show the Required property.
   Debug.Print "Fields in " & tdfTemp.Name & ":"
   For Each fldLoop In tdfTemp.Fields
      Debug.Print , fldLoop.Name & ", Required = " & _
         fldLoop.Required
   Next fldLoop

End Sub
```

StillExecuting Property and Cancel Method Example

This example uses the **StillExecuting** property and the **Cancel** method to asynchronously open a **Connection** object.

```
Sub CancelConnectionX()

   Dim wrkMain As Workspace
   Dim conMain As Connection
   Dim sngTime As Single

   Set wrkMain = CreateWorkspace("ODBCWorkspace", _
      "admin", "", dbUseODBC)
   ' Open the connection asynchronously.
   Set conMain = wrkMain.OpenConnection("Publishers", _
      dbDriverNoPrompt + dbRunAsync, False, _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")

   sngTime = Timer

   ' Wait five seconds.
   Do While Timer - sngTime < 5
   Loop

   ' If the connection has not been made, ask the user
   ' if she wants to keep waiting. If she does not, cancel
   ' the connection and exit the procedure.
   Do While conMain.StillExecuting

      If MsgBox("No connection yet--keep waiting?", _
            vbYesNo) = vbNo Then
         conMain.Cancel
         MsgBox "Connection cancelled!"
         wrkMain.Close
         Exit Sub
      End If

   Loop

   With conMain
      ' Use the Connection object conMain.
      .Close
   End With

   wrkMain.Close

End Sub
```

This example uses the **StillExecuting** property and the **Cancel** method to asynchronously execute a **QueryDef** object.

```
Sub CancelQueryDefX()

   Dim wrkMain As Workspace
   Dim conMain As Connection
   Dim qdfTemp As QueryDef
   Dim sngTime As Single
```

```
    Set wrkMain = CreateWorkspace("ODBCWorkspace", _
        "admin", "", dbUseODBC)
    Set conMain = wrkMain.OpenConnection("Publishers", _
        dbDriverNoPrompt, False, _
        "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")

    Set qdfTemp = conMain.CreateQueryDef("")

    With qdfTemp
        .SQL = "UPDATE roysched " & _
            "SET royalty = royalty * 2 " & _
            "WHERE title_id LIKE 'BU____' OR " & _
            "title_id LIKE 'PC____'"

        ' Execute the query asynchronously.
        .Execute dbRunAsync

        sngTime = Timer

        ' Wait five seconds.
        Do While Timer - sngTime < 5
        Loop

        ' If the query has not completed, ask the user if
        ' she wants to keep waiting. If she does not, cancel
        ' the query and exit the procedure.
        Do While .StillExecuting

            If MsgBox( _
                    "Query still running--keep waiting?", _
                    vbYesNo) = vbNo Then
                .Cancel
                MsgBox "Query cancelled!"
                Exit Do
            End If

        Loop

    End With

    conMain.Close
    wrkMain.Close

End Sub
```

This example uses the **StillExecuting** property and the **Cancel** method to asynchronously move to the last record of a **Recordset** object.

```
Sub CancelRecordsetX()

    Dim wrkMain As Workspace
    Dim conMain As Connection
    Dim rstTemp As Recordset
    Dim sngTime As Single

    Set wrkMain = CreateWorkspace("ODBCWorkspace", _
        "admin", "", dbUseODBC)
```

```
   Set conMain = wrkMain.OpenConnection("Publishers", _
      dbDriverNoPrompt, False, _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")
   Set rstTemp = conMain.OpenRecordset( _
      "SELECT * FROM roysched", dbOpenDynaset)

   With rstTemp

      ' Call the MoveLast method asynchronously.
      .MoveLast dbRunAsync

      sngTime = Timer

      ' Wait five seconds.
      Do While Timer - sngTime < 5
      Loop

      ' If the MoveLast has not completed, ask the user if
      ' she wants to keep waiting. If she does not, cancel
      ' the MoveLast and exit the procedure.
      Do While .StillExecuting

         If MsgBox( _
               "Not at last record yet--keep waiting?", _
               vbYesNo) = vbNo Then
            .Cancel
            MsgBox "MoveLast cancelled!"
            conMain.Close
            wrkMain.Close
            Exit Sub
         End If

      Loop

      ' Use recordset.

      .Close

   End With

   conMain.Close
   wrkMain.Close

End Sub
```

Transactions Property Example

This example demonstrates the **Transactions** property in Microsoft Jet and ODBCDirect workspaces.

```
Sub TransactionsX()

   Dim wrkJet As Workspace
   Dim wrkODBC As Workspace
   Dim dbsNorthwind As Database
   Dim conPubs As Connection
   Dim rstTemp As Recordset

   ' Open Microsoft Jet and ODBCDirect workspaces, a Microsoft
   ' Jet database, and an ODBCDirect connection.
   Set wrkJet = CreateWorkspace("", "admin", "", dbUseJet)
   Set wrkODBC = CreateWorkspace("", "admin", "", dbUseODBC)
   Set dbsNorthwind = wrkJet.OpenDatabase("Northwind.mdb")
   Set conPubs = wrkODBC.OpenConnection("", , , _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")

   ' Open two different Recordset objects and display the
   ' Transactions property of each.

   Debug.Print "Opening Microsoft Jet table-type " & _
      "recordset..."
   Set rstTemp = dbsNorthwind.OpenRecordset( _
      "Employees", dbOpenTable)
   Debug.Print "    Transactions = " & rstTemp.Transactions

   Debug.Print "Opening forward-only-type " & _
      "recordset where the source is an SQL statement..."
   Set rstTemp = dbsNorthwind.OpenRecordset( _
      "SELECT * FROM Employees", dbOpenForwardOnly)
   Debug.Print "    Transactions = " & rstTemp.Transactions

   ' Display Transactions property of a Connection object in
   ' an ODBCDirect workspace.
   Debug.Print "Testing Transaction property of " & _
      "an ODBC connection..."
   Debug.Print "    Transactions = " & conPubs.Transactions

   rstTemp.Close
   dbsNorthwind.Close
   conPubs.Close
   wrkJet.Close
   wrkODBC.Close

End Sub
```

UserName Property Example

This example uses the **UserName** property to change a particular user's permissions on an object and to verify that user's ability to append new data to the same object.

```
Sub UserNameX()

   ' Ensure that the Microsoft Jet workgroup information
   ' file is available.
   DBEngine.SystemDB = "system.mdw"

   Dim dbsNorthwind As Database
   Dim docTemp As Document

   Set dbsNorthwind = OpenDatabase("Northwind.mdb")

   Set docTemp = _
      dbsNorthwind.Containers("Tables").Documents(0)

   ' Change the permissions of NewUser on the first Document
   ' object in the Tables container.
   With docTemp
      .UserName = "NewUser"
      .Permissions = dbSecRetrieveData
      If (.Permissions And dbSecInsertData) = _
            dbSecInsertData Then
         Debug.Print .UserName & " can insert data."
      Else
         Debug.Print .UserName & " can't insert data."
      End If
   End With

   dbsNorthwind.Close

End Sub
```

Version Property Example

This example uses the **Version** property to report on the Microsoft Jet database engine in memory, a Microsoft Jet database, and an ODBC connection.

```
Sub VersionX()

   Dim wrkJet As Workspace
   Dim dbsNorthwind As Database
   Dim wrkODBC As Workspace
   Dim conPubs As Connection

   ' Open Microsoft Jet Database object.
   Set wrkJet = CreateWorkspace("NewJetWorkspace", _
      "admin", "", dbUseJet)
   Set dbsNorthwind = wrkJet.OpenDatabase("Northwind.mdb")

   ' Create ODBCDirect Workspace object and open Connection
   ' objects.
   Set wrkODBC = CreateWorkspace("NewODBCWorkspace", _
      "admin", "", dbUseODBC)
   Set conPubs = wrkODBC.OpenConnection("Connection1", , , _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers")

   ' Show three different uses for the Version property.
   Debug.Print "Version of DBEngine (Microsoft Jet " & _
      "in memory) = " & DBEngine.Version
   Debug.Print "Version of the Microsoft Jet engine " & _
      "with which " & dbsNorthwind.Name & _
      " was created = " & dbsNorthwind.Version
   Debug.Print "Version of ODBCDirect connection " & _
      "(using Database property) = " & _
      conPubs.Database.Version

   dbsNorthwind.Close
   conPubs.Close
   wrkJet.Close
   wrkODBC.Close

End Sub
```

Connect and ReturnsRecords Properties Example (Client/Server)

This example uses the **Connect** and **ReturnsRecords** properties to select the top five book titles from a Microsoft SQL Server database based on year-to-date sales amounts. In the event of an exact match in sales amounts, the example increases the size of the list displaying the results of the query and prints a message explaining why this occurred.

```
Sub ClientServerX1()

   Dim dbsCurrent As Database
   Dim qdfPassThrough As QueryDef
   Dim qdfLocal As QueryDef
   Dim rstTopFive As Recordset
   Dim strMessage As String

   ' Open a database from which QueryDef objects can be
   ' created.
   Set dbsCurrent = OpenDatabase("DB1.mdb")

   ' Create a pass-through query to retrieve data from
   ' a Microsoft SQL Server database.
   Set qdfPassThrough = _
      dbsCurrent.CreateQueryDef("AllTitles")
   qdfPassThrough.Connect = _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers"
   qdfPassThrough.SQL = "SELECT * FROM titles " & _
      "ORDER BY ytd_sales DESC"
   qdfPassThrough.ReturnsRecords = True

   ' Create a temporary QueryDef object to retrieve
   ' data from the pass-through query.
   Set qdfLocal = dbsCurrent.CreateQueryDef("")
   qdfLocal.SQL = "SELECT TOP 5 title FROM AllTitles"

   Set rstTopFive = qdfLocal.OpenRecordset()

   ' Display results of queries.
   With rstTopFive
      strMessage = _
         "Our top 5 best-selling books are:" & vbCr

      Do While Not .EOF
         strMessage = strMessage & "     " & !Title & _
            vbCr
         .MoveNext
      Loop

      If .RecordCount > 5 Then
         strMessage = strMessage & _
            "(There was a tie, resulting in " & _
            vbCr & .RecordCount & _
            " books in the list.)"
      End If

      MsgBox strMessage
      .Close
   End With
```

```
    ' Delete new pass-through query because this is a
    ' demonstration.
    dbsCurrent.QueryDefs.Delete "AllTitles"
    dbsCurrent.Close

End Sub
```

CreateQueryDef Method, OpenRecordset Method, and SQL Property Example (Client/Server)

This example uses the **CreateQueryDef** and **OpenRecordset** methods and the **SQL** property to query the table of titles in the Microsoft SQL Server sample database Pubs and return the title and title identifier of the best-selling book. The example then queries the table of authors and instructs the user to send a bonus check to each author based on his or her royalty share (the total bonus is $1,000 and each author should receive a percentage of that amount).

```
Sub ClientServerX2()

   Dim dbsCurrent As Database
   Dim qdfBestSellers As QueryDef
   Dim qdfBonusEarners As QueryDef
   Dim rstTopSeller As Recordset
   Dim rstBonusRecipients As Recordset
   Dim strAuthorList As String

   ' Open a database from which QueryDef objects can be
   ' created.
   Set dbsCurrent = OpenDatabase("DB1.mdb")

   ' Create a temporary QueryDef object to retrieve
   ' data from a Microsoft SQL Server database.
   Set qdfBestSellers = dbsCurrent.CreateQueryDef("")
   With qdfBestSellers
      .Connect = "ODBC;DATABASE=pubs;UID=sa;PWD=;" & _
          "DSN=Publishers"
      .SQL = "SELECT title, title_id FROM titles " & _
         "ORDER BY ytd_sales DESC"
      Set rstTopSeller = .OpenRecordset()
      rstTopSeller.MoveFirst
   End With

   ' Create a temporary QueryDef to retrieve data from
   ' a Microsoft SQL Server database based on the results from
   ' the first query.
   Set qdfBonusEarners = dbsCurrent.CreateQueryDef("")
   With qdfBonusEarners
      .Connect = "ODBC;DATABASE=pubs;UID=sa;PWD=;" & _
         "DSN=Publishers"
      .SQL = "SELECT * FROM titleauthor " & _
         "WHERE title_id = '" & _
         rstTopSeller!title_id & "'"
      Set rstBonusRecipients = .OpenRecordset()
   End With

   ' Build the output string.
   With rstBonusRecipients
      Do While Not .EOF
         strAuthorList = strAuthorList & "    " & _
            !au_id & ":  $" & (10 * !royaltyper) & vbCr
         .MoveNext
      Loop
   End With

   ' Display results.
```

```
   MsgBox "Please send a check to the following " & _
      "authors in the amounts shown:" & vbCr & _
      strAuthorList & "for outstanding sales of " & _
      rstTopSeller!Title & "."

   rstTopSeller.Close
   dbsCurrent.Close

End Sub
```

CreateTableDef Method, FillCache Method, and CacheSize, CacheStart and SourceTableName Properties Example (Client/Server)

This example uses the **CreateTableDef** and **FillCache** methods and the **CacheSize**, **CacheStart** and **SourceTableName** properties to enumerate the records in a linked table twice. Then it enumerates the records twice with a 50-record cache. The example then displays the performance statistics for the uncached and cached runs through the linked table.

```
Sub ClientServerX3()

   Dim dbsCurrent As Database
   Dim tdfRoyalties As TableDef
   Dim rstRemote As Recordset
   Dim sngStart As Single
   Dim sngEnd As Single
   Dim sngNoCache As Single
   Dim sngCache As Single
   Dim intLoop As Integer
   Dim strTemp As String
   Dim intRecords As Integer

   ' Open a database to which a linked table can be
   ' appended.
   Set dbsCurrent = OpenDatabase("DB1.mdb")

   ' Create a linked table that connects to a Microsoft SQL
   ' Server database.
   Set tdfRoyalties = _
      dbsCurrent.CreateTableDef("Royalties")
   tdfRoyalties.Connect = _
      "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=Publishers"
   tdfRoyalties.SourceTableName = "roysched"
   dbsCurrent.TableDefs.Append tdfRoyalties
   Set rstRemote = _
      dbsCurrent.OpenRecordset("Royalties")

   With rstRemote
      ' Enumerate the Recordset object twice and record
      ' the elapsed time.
      sngStart = Timer

      For intLoop = 1 To 2
         .MoveFirst
         Do While Not .EOF
            ' Execute a simple operation for the
            ' performance test.
            strTemp = !title_id
            .MoveNext
         Loop
      Next intLoop

      sngEnd = Timer
      sngNoCache = sngEnd - sngStart

      ' Cache the first 50 records.
      .MoveFirst
```

```
      .CacheSize = 50
      .FillCache
      sngStart = Timer

      ' Enumerate the Recordset object twice and record
      ' the elapsed time.
      For intLoop = 1 To 2
         intRecords = 0
         .MoveFirst
         Do While Not .EOF
            ' Execute a simple operation for the
            ' performance test.
            strTemp = !title_id
            ' Count the records. If the end of the
            ' cache is reached, reset the cache to the
            ' next 50 records.
            intRecords = intRecords + 1
            .MoveNext
            If intRecords Mod 50 = 0 Then
               .CacheStart = .Bookmark
               .FillCache
            End If
         Loop
      Next intLoop

      sngEnd = Timer
      sngCache = sngEnd - sngStart

      ' Display performance results.
      MsgBox "Caching Performance Results:" & vbCr & _
         "    No cache: " & Format(sngNoCache, _
         "##0.000") & " seconds" & vbCr & _
         "    50-record cache: " & Format(sngCache, _
         "##0.000") & " seconds"
      .Close
   End With

   ' Delete linked table because this is a demonstration.
   dbsCurrent.TableDefs.Delete tdfRoyalties.Name
   dbsCurrent.Close

End Sub
```