ADOxtra Documentation

www.xtramania.com

ADOxtra usage

About

What is ADOxtra?

ADOxtra is an extension of Macromedia Director of type 'scripting xtra'. It extends the capabilities of Lingo (Director's scripting language) with ability to access databases using ADO.

ADOxtra allows you to retrieve or modify data in MS Access files, MS SQL Server and Oracle databases, ODBC compliant databases from authoring or projector locally and over LAN or from Shockwave. It is capable to access databases over Internet.

ADOxtra provides recordset based data accessing with automatic type casting that grants the ability to work with a database field value of those type that it is in a database. So integer database fields look like integer Lingo values and text database fields look like string Lingo values, etc. You do not need to explicitly convert values.

What is ADO?

ADO (Active Data Objects) component is one of Microsoft Data Access Components (MDAC).

MDAC is a modern system level extension from Microsoft providing database management capabilities. It is already installed with Windows 98/ME/2000, Internet Explorer, Microsoft Office and other software.

It is already present on the most of user systems.

Installers for Windows 95/NT systems are available online at http://www.microsoft.com/data/download.htm.

The complete ADO documentation is available online at http://msdn.microsoft.com/library/en-us/ado270/htm/dasdkadooverview.asp.

Also see http://www.microsoft.com/data/ for more information about MDAC.

ADO object model

ADO (ActiveX Data Objects) are a set of objects that provides the high level interface to OLE DB database providers. Database providers may perform actual database accessing or may serve as lower level interface to other databasing components. MDAC (Microsoft Data Access Components) contain several OLE DB providers for Access, MS SQL, Oracle, ODBC and several others. Also MDAC contain Microsoft Remoting provider. It is used to provide remote (over Internet) connection to the database. MS Remote provider is used on the client side and one of the other providers is used on the server side.

So, the one of the basic points of using ADO is specifying correct connection parameters which have to define the provider, the data source and other information specific to the provider. ADO provides the Connection object that is responsible for the establishing connection to some provider. Also Connection object is responsible for the general connection operations like transactions management.

One of the most important objects which ADO consists of is the Recordset object. Recordset object serves as an interface to the actual database data. Recordset can access the data one record at a time. The current record is represented via collection of Field objects, where each Field object corresponds to the field of a table or data query. Recordset object provides browsing capabilities to move through the set of records. Some of Recordset's capabilities depends on the provider, others depends on the settings of certain properties.

In general, there are several ways how to get connected to the database. The simple scenario includes: creating a recordset object, invoking Recordset's Open method with parameters providing minimum amount of information. The advanced scenario usually includes more steps: creating connection object, specifying multiple connection options, opening connection, creating recordset object, binding recordset to the opened connection, adjusting recordset properties and opening recordset. In most cases it is enough to use simple scenario since the default ADO behavior is often just what you need.

Common Guidelines

ADOxtra emulates VB syntax as close as possible

ADOxtra extends the functionality of Lingo to allow you to use ADO right from Lingo. ADO is implemented via COM Automation technology and was designed mostly for Visual Basic clients. VB syntax is native for ADO. ADOxtra tries to emulate Vb syntax as close as possible with Lingo. See more details about differences between real VB syntax and Lingo syntax using ADOxtra in reference section.

ADOxtra provides wrapper instances for every ADO object

ADOxtra extensively uses Lingo's dot syntax to allow cascading access to ADO object's methods and properties. VbScriptXtra provides its own wrapper instance for every ADO object reference. You may think about such wrappers as just Lingo value of some special type. Any method or property access of the automation object is implemented through this wrapper.

`CreateObject` method is the main starting point while using ADOxtra. Like its Visual Basic analogue, this method creates a new instance of the ADO object and returns it as a Lingo value. This Lingo value is provided by ADOxtra to allow direct access to the object's methods and properties.

Use `CreateObject` to start databasing

`CreateObject` is implemented by ADOxtra as xtra-level method. So it can be called only in xtra context. This is done intentionally to avoid possible conflicts with other xtras which may use the same method's name. So Lingo allows you to either place xtra reference as the first argument of the method or use dot syntax. Following lines does the same:

```
rst=CreateObject(xtra "ADOxtra",#Recordset)
```

or

```
rst=xtra("ADOxtra").CreateObject(#Recordset)
```

The returned Lingo value is actually an ADOxtra instance, which wraps an ADO object. This ADOxtra wrapper passes onto wrapped ADO object corresponding method and property calls making all necessary type casting.

Several notes, before you begin

ADO objects use a large amount of constant values (enumeration constants) as values of properties and function parameters. For example, `Recordset.LockType` property may accept several numeric values that indicate the type of locking to be applied to the recordset. This numeric values have their own names (like `adLockReadOnly` which equals 1). You may use numeric values directly or make ADOxtra to convert constant name to the corresponding value. Every ADOxtra object wrapper has this capability. Just

use the name of constant value as usual object's property:
`rst.CursorType=rst.adOpenKeyset`

It is the same as:
`rst.CursorType=1`

Also, many ADO parameters or properties may accept or return a bitmask of constant values. Use Lingo bitwise operations to set or get necessary information (`bitOr(arg1,arg2), bitAnd(arg1,arg2)`).

Several ADO objects' methods may use optional arguments. To skip optional argument at the end of argument list you may just omit it. To skip optional argument in the middle of the argument list use `VOID` instead. For example:

```
rst.Open("SomeTable",void,void,void,rst.adCmdTable)
```

ADO objects use several collections. Collection is just a list of items. For example, Recordset object contains a collection of Field objects. In Visual Basic you may use several approaches to get the certain item in collection. The most common way is Item(index) method of the collection. ADOxtra does not support Item() method due to some syntax limitations in Director 7. Instead, ADOxtra offers square brackets [] operator to access an item of the collection. For example, use:

```
put rst.Fields["Name"] -- getting the default Value property
put rst.Fields["Name"].Type -- getting the type property
```

to access collection's items. Another collection supported by ADOxtra is `Properties` collection of `Connection` and `Recordset` objects. Also ADOxtra has limited support for `Connection.Errors` collection

Simple databasing scenario

Initialization ADOxtra

Usually, you begin with `Init` method of the ADOxtra. It makes COM initialization and allows you to set default wrapper objects mode to debug mode. When debug mode is set, all ADOxtra objects output their error information to Director or projector Message window. It is highly recommended to set the debug property to true when you just play with the xtra. So, use:

```
bSuccess=Init(xtra"ADOxtra",true) -- Initializing xtra and setting
debug mode
```

Creating recordset object

After successful call of `Init` method, you are ready to create Recordset object. Use ADOxtra function `CreateObject()` with parameter `#Recordset` to create wrapper for ADODB.Recordset object:

```
  rst=CreateObject(xtra"ADOxtra",#Recordset)
```

Check resulting value to ensure that ADO is available. If function succeeded `rst` will be the Lingo object reference, otherwise it will be a string, describing error:

```
if objectP(rst) then
  put "Recordset created"
else
  put "Error:"&&rst
end if
```

Choosing which database to open

ADO usually uses a `connection string` to specify to which database to connect or which database to open. Connection string is the string in form "PropertyName=PropertyValue;OtherPropertyName=OtherValue". Use `BuildConnectionString` ADOxtra xtra-level method to invoke a standard dialog for building connection string. Here is several samples, how the connection string may look like:

MS Access databases

```
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=D:\Temp\DB.mdb; Mode=ReadWrite"
```

MS Access password protected databases

```
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=D:\Temp\DB.mdb; Mode=ReadWrite;
Jet OLEDB:Database Password=PasswordHere"
```

MS Access databases via ODBC driver (DSNless connection):

```
connectionString="DRIVER={Microsoft Access Driver (*.mdb)};
DBQ=D:\Temp\DB.mdb"
```

MS SQL Server:

```
connectionString="Provider=SQLOLEDB.1; Integrated Security=SSPI;
Persist Security Info=False; Initial Catalog=DemoDB;
Data Source=SqlServerName"
```

Oracle databases:

```
connectionString="Provider=MSDAORA.1; Password=psw; User ID=admin;
Data Source=srv; Persist Security Info=True"
```

The most important property in connection string is "Provider". Its value usually determines the type of database to work with. Other properties specify additional information that may be specific to the provider. Note that if you omit the provider property, the default will be used. Default provider for ADO is OLE DB Provider for ODBC as in the second example above.

Note that connection string may specify the type of access to data. In the first example "Mode=ReadWrite" specifies that connection to database is for reading and writing. All or almost all information specified in connection string may be adjusted directly by setting properties of the connection object. But in simple scenario you do not use Connection object directly, although ADO will create it implicitly during processing of the recordset's Open method. So, in simple scenario connection string is the only source of information about which database to open.

Opening recordset object

To get actual database data with ADO you have to open a recordset with specified command text over specified connection. The command text may be a SQL query or command, a table name, a stored procedure name, or other provider specific command.

To open recordset you may call the recordset's `Open` method:

```
connectionString="Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=D:\Temp\DB.mdb; Mode=ReadWrite;"
source="SELECT SomeFieldName, SomeOtherFieldName FROM SomeTable
ORDER BY SomeFieldName"
rst.Open(source,connectionString)
if rst.succeeded then
  put "Recordset state:"&&rst.State
else
  put "Error:"&&rst.lastError
end if
```

Be sure to always check whether call was successful if you do not use ADOxtra's debugMode, since ADO often (but not always) return useful error descriptions, if you do something incorrectly. After `Open` call succeeded check the state property of the recordset. Usually if source text specifies row-returning query (like SELECT), the `rst.state` property will be set to `adStateOpen` (=1). If source text specifies command query (like INSERT), the state of recordset object will be set to `adStateClosed` (=0).

Getting data from database via recordset

The recordset object with `rst.state = adStateOpen` is ready to provide access to the data. Recordset provides access to the data in record by record manner. So at any given moment you can only access the current record. Move the current record of a recordset with `rst.MoveNext()`, `rst.MovePrevious()`, `rst.MoveFirst`, `rst.MoveLast()` functions. Use `rst.EOF` and `rst.BOF` properties to determine whether recordset has reached the end or the beginning. Use `rst.Fields` collection to actually work with data:

```
repeat while not rst.eof
  put rst.fields["SomeFieldName" or SomeFieldIndex]
  rst.MoveNext()
end repeat
```

See more info about `Fields` collection and `Field` object to find out what else you can do with them. Also you may use `rst.GetString` method to quickly see the string representation of the recordset data. `GetString` method lets you specify custom strings for field delimiters, row delimiters and `Null` expression.

Modifying data via recordset

By default, recordset's `Open` method will open read only forward only recordset. It means such recordset will not be able to modify data and will not be able to move the current record backward. This behavior is determined by other parameters of `rst.Open` method. See the description of `cursorType` and `lockType` parameters of `rst.Open` method. In general, `lockType` parameter determines the type of locking to be applied to the data. The default value is `adLockReadOnly`, which allows only read access to the data. The `cursorType` defines the capabilities of the recordset in relation to data changes made by others. The default value is `adOpenForwardOpen`, which defines a static copy of a set of records with forward only movement capability. Usually, in case you are going to modify data in database you may set the `lockType` parameter to `adLockPessimistic` and the `cursorType` parameter to `adOpenKeyset`:

```
rst.Open(source,connectionString,rst.adOpenKeyset,rst.adLockPessim
istic)
if rst.succeeded then
  put "Recordset state:"&&rst.State
else
  put "Error:"&&rst.lastError
end if
```

Now you are able to make modifications to data:

```
rst.Fields["SomeFieldName"]=SomeNewValue
rst.Fields["SomeOtherFieldName"]=SomeOtherNewValue
rst.Update()
```

The actual data modification is occurred on `Update` method. Always check whether call was succeeded, since data provider may deny attempt to modify data if data violates database integrity or other database rules.

Closing recordset

After you finish using particular recordset you may reopen it with other parameters. Use `rst.Close` method to release system resources associated with open recordset. Then you may reopen it with other parameters. If you do not need it any more, be sure to void out any Lingo variable that may store a reference to the ADOxtra wrapper object, thus completely releasing it from memory.

Some advanced details

Connection object

In certain cases you may need to use alternative approach to perform required task. For example, you have to create connection object before opening recordset to open recordset inside a transaction. The other example is retrieving database schema information.

Use ADOxtra function CreateObject() with parameter #Connection to create wrapper for ADODB.Connection object:

```
cnn=CreateObject(xtra"ADOxtra",#Connection)
```

Check resulting value to ensure that ADO is available. If function succeeded `cnn` will be the Lingo object reference, otherwise it will be a string, describing error. Use `cnn.Version` property to determine ADO version:

```
if objectP(cnn) then
  put "ADO version:"&&cnn.Version
else
  put "Error:"&&db
end if
```

Then you have to adjust connection parameters using Connection object's properties. See `cnn.ConnectionString`, `cnn.Provider` and other properties of the `Connection` object. Otherwise you may specify connection information as parameters of `cnn.Open` method. It is definitely good idea do not specify the same kind of information twice, since different versions of ADO may behave differently, deciding which information to take into account. For example, if you set `cnn.Provider` property and set alternative provider info in the `cnn.ConnectionString` property, different versions of ADO may try to use different providers, probably generating an error in one of the cases.

Object's dynamic properties

Connection object contains the collection of dynamic properties `cnn.Properties`. This collection contains multiple properties specific to the provider. You may access this collection after you specify which provider to use. If you do not specify any, the OLE DB provider for ODBC will be used. Once you set the provider of the connection object you cannot change it for this particular instance. After you specify provider you may look at dynamic properties it supports:

```
cnn.Provider="Microsoft.Jet.OLEDB.4.0"
repeat with i = 0 to cnn.Properties.Count - 1
  put cnn.Properties[i].Name && "=" && cnn.Properties[i]
end repeat
```

See more details about property object here. You may adjust some dynamic properties:

```
cnn.Properties["SomePropertyName"]=SomeNewPropertyValue
```

The recordset object contains its own provider specific collection of the dynamic properties. They may be accessed the same way.

Using transactions

You may use opened connection to start transaction. Use `cnn.BeginTrans` to start transaction. Use `cnn.CommitTrans` method to save changes or `cnn.RollbackTrans` method to cancel the changes being made inside the current transaction.

Mapping between Lingo data types and ADO data types

ADOxtra performs automatic type casting to correctly transfer data between Lingo and ADO object and vice versa. Typecasting operations are implicitly performed by ADOxtra while processing Lingo method arguments, returning values, and property values.

ADO and Visual Basic supports rather large amount of data types. Lingo has its own Director specific data types. So, ADOxtra may not find suitable conversion in all cases, although it provides conversion in the most cases. If ADOxtra does not know how to convert the value it will report an error.

See detailed description which types are mapped to which types in Reference section.

Debugging your scripts using ADOxtra

Every ADOxtra wrapper instance has internal last error flag and error description. The last error flag is cleared before any access to the ADO object. If property access or method call failed or another error happened, this flag is raised. So you may detect whether last call completed successfully. Use `object.Failed` or `object.Succeeded` properties to check whether the last call was successfull. If an error happened you may see its description using `object.lastError` property.

Also you may adjust ADOxtra wrapper objects to output its `lastError` directly to the Message window every time error happens. Set `object.DebugMode` to true (1) to do this. By default, ADOxtra wrappers created by `CreateObject` call inherit the xtra's default value for debugMode. You may change this default with `Init` xtra-level method.

Note: While you are just investigating ADO databasing capabilities it is better to set Debug Mode by default, to ensure you always know if something goes wrong.

Remote databasing

Accessing databases over Internet and from Shockwave

ADOxtra may be used to access databases over Internet right from usual Director movie or Shockwave movie.

MDAC (Microsoft Data Access Components) contain MS Remoting Provider (Provider=MS Remote) which is capable to retrieve or send recordset data over Internet via http or https protocols. MDAC have to be installed (usually it is already installed) on both client and server sides. Remoting provider is used on the client side. It connects to the server part via http (https) protocol. The server side has to be running IIS web site. IIS provides ISAPI extensions mechanism, which allows MS Remoting Provider to call its counter part on the server. IIS web site and server itself have to be properly configured to allow remote databasing. Also there are several server security issues to take into account.

Note, ADOxtra is limited to remote only databasing while it is used from Shockwave movie. This limitation is done intentionally to conform to Shockwave safety requirements.

Server configuration and security for remote databasing

At first, web server used for remote databasing must contain virtual directory MSADC with "execute programs" enabled. This virtual directory has to contain `msadcs.dll` file. Copy this file from `C:\Program Files\Common Files\System\msadc`.

Server security may need to be adjusted to allow remote databasing. By default, ADO is installed with a "safe" server configuration. The registry file `handsafe.reg` (`C:\Program Files\Common Files\System\msadc`) has been provided to set up the handler registry entries for a safe configuration. To ensure you your server runs in a safe mode, run `handsafe.reg`. The registry file `handunsf.reg` has been provided to set up the handler registry entries for an unrestricted configuration. To run in unrestricted mode, run `handunsf.reg`.

Safe configuration uses `Msdfmap.ini` file located in Windows (probably WinNT) directory. You must configure this file according to your needs, before using remote databasing. By default this file denies all remote connections to databases. `Msdfmap.ini` file may specify valid data sources that are allowed to access remotely, named SQL queries, access rights etc. See MSDN library to get more information about this file or look at the file itself since it contains necessary information and samples.

To get remote access to the database you have to be able to access database locally on the web server's computer. Note that IIS may execute server part of RDS under IIS's Internet user account (IUSR_ComputerName). This may be important when connecting to MS SQL Server or databases over LAN.

The typical `msdfmap.ini` file may look like:

```
[connect default]
```

```
;If we want to disable unknown connect values, we set Access to
NoAccess
Access=NoAccess
[connect YourDataSourceName]
Access=ReadWrite
Connect="YourConnectionStringHereToBeUsedOnServer"
```

This data source name may be used in connection string specified by client. Furthermore, this is the only valid data source name in this example, since all remote connections other then `YourDataSourceName` will be blocked by `[connect default]` section. To provide even more security, you may also define named SQL queries, which will be the only allowed SQL queries from remote clients.

Client side

At client side (it is your usual Director movie or Shockwave movie somewhere in the world, that is running on the machine connected to the Internet) you have to specify the correct connection parameters. You have to specify that you are going to use MS Remote provider to connect to your web site via http and your web server will use the data source name specified in server's `Msdfmap.ini` or some other provider to access some database. See examples below. Also you may set Internet Timeout property. It defines how long client will wait for the answer in milliseconds. Default setting is 5 minutes.

Note, the above sample of `msdfmap.ini` file allows you to only use connection string below, since other data sources will be blocked by `Access=NoAccess` statement of the `[Connect Default]` section:

Using data source name specified in `msdfmap.ini` file on web server:

```
connectionString="Provider=MS Remote;
Data Source=YourDataSourceName;
Remote Server=http://YourWebServer; Internet Timeout=300000"
```

In this case the actual connection string is contained in `[connect YourDataSourceName]` section of the `Msdfmap.ini` file (See above).

If you do not worry about security you may allow default remote access to databases through your web server. Just set required access type in Access=ReadWrite statement of the `[Connect Default]` section. If default access is enabled you may use 'fully qualified' connection strings on the client side:

MS Access databases over Internet:

```
connectionString="Provider=MS Remote; Data Source=D:\Demo\DB.mdb;
Mode=Read; Remote Server=http://YourWebServer; Remote
Provider=Microsoft.Jet.OLEDB.4.0; Internet Timeout=300000"
```

MS SQL Server:

```
connectionString="Provider=MS Remote; Data Source=SqlServerName;
Initial Catalog=DemoDB; Remote Server=http://YourWebServer; Remote
Provider=SQLOLEDB.1; Internet Timeout=300000"
```

ODBC system DSN:

```
connectionString="Provider=MS Remote; Data Source=YourSystemDSN;
Remote Server=http://YourWebServer; Remote Provider=MSDASQL.1;
Internet Timeout=300000"
```

After setting correct connection string you may use Connection or Recordset objects as usual. Note that real data transfer occurs when you open recordset or update it.

Director settings for autodownloading ADOxtra from Shockwave

To use ADOxtra from Shockwave several extra steps are required. At first you have to place the ADOxtra package (certified by Verisign) on your web site. Otherwise you may use package available at http://download.adoxtra.com/package/. Then you have to modify xtrainfo.txt file located in you Director installation folder. This file provides the master list of xtras known to Director. Add following lines to the end of this file:

[#nameW32:"ADOxtra.x32", #info:"http://www.adoxtra.com/",
#package:"http://download.adoxtra.com/package/ADOxtra"]

[#nameW32:"ADOxtraLite.x32", #info:"http://www.adoxtra.com/",
#package:"http://download.adoxtra.com/package/ADOxtraLite"]

Replace the #package values to the location on your web server where package file resides. Note: file and folder names in URL may be case sensitive. After modifying this file you may run Director, open the movie you are going to use from Shockwave and then invoke the Modify/Movie/Xtras dialog. Find (or add) the ADOxtra (ADOxtraLite) entries and place the check mark near "Download if needed" and "Include in projector". Director will check whether package file is available at this moment. Then you may save the movie as Shockwave and try it.

Frequently Asked Questions about ADOxtra

Which xtra is better: ADOxtra or VbScriptXtra?

Although both xtras allows using ADO in Macromedia Director, they are quite different.

Consider following notes about these xtras:

1. ADOxtra is safe for Shockwave. VbScriptXtra is not safe for Shockwave.

2. ADOxtra is implemented using 'compilation time binding' with ADO objects, while VbScriptXtra uses run-time 'late binding'. So, in general, ADOxtra is a bit faster than VbScriptXtra, although this difference is not important since actual data accessing usually takes much more time.

3. But, due to the same reason, ADOxtra supports a rather limited subset of ADO, while VbScriptXtra supports almost anything currently available and probably future ADO extensions.

ADOxtra supports ADO interfaces of version 2.0 with Connection, Recordset, Field, Property and Error objects. This subset is more than enough in the most of cases required for Macromedia Director.

But ADO provides much more functionality. There are also other ADO-friendly components like ADOX - ADO extensions for database management, ADOMD - multi dimensional databases.

All these components are available with VbScriptXtra. Also you can use DAO with VbScriptXtra.

4. The price is almost the same…

Can ADOxtra connect to mySQL database?

ADO (and therefore ADOxtra) can connect to any database via ODBC driver, if one is installed. MySQL provides an ODBC driver MyODBC, which allows using MySQL databases via ADO.

Note also, MyODBC is not installed on usual user's system. That is why there is probably no sense to use it from Shockwave... So remote databasing using ADO is actually possible with IIS web server only.

But it could be used from usual Director movie which can install appropriate ODBC driver to a user's system.

MyODBC driver is available at www.mysql.com

ADOxtra Reference

Version history

April 17th, 2001

Weakened: Demo limitations has been weakened. Now the demo version is fully functional while the current record is within the first 24 records of a recordset. If the current record is outside this range, `MoveNext`, `MovePrevious`, setting `AbsolutePosition` and `AbsolutePage` recordset's properties may fail with "Demo version limitation" error.

Applied: a workaround recommended by Microsoft's knowledge base to avoid possible `rst.GetString()` failures with ADO 2.0, 2.1.

Added: Several properties of Recordset object: `PageCount`, `PageSize`, `AbsolutePage`, `AbsolutePosition`.

Added: Support for "numeric" data type which is used in different databases.

March 14th, 2001

Bug fix: Delete method of recordset object cannot be called due to Director limitation (Director just does not call xtra implementation of the Delete method). Use `rst.Delete_()` to call Delete method of the wrapped ADODB.Recordset object.

Added: Xtra level function `BuildConnectionString`. It is used to invoke a dialog for choosing data source and building connection string.

Added: Global functions for date/time conversion from float representation to Lingo property list and vice versa. (`DateTimeListToFloat` and `FloatToDateTimeList`).

Added: Multiple enumeration constants used by ADO now available as usual properties of every wrapper object. For example: `cnn.adAsyncFetch`

Added: Shockwave is supported now. While used from Shockwave, MS Remote is the only supported provider due to Shockwave safety reasons. Attempts to use other providers generate error: "Provider is not supported from Shockwave."

Added: Multiple methods and properties defined in ADO 2.0 (`rst.GetString()`, `rst.UpdateBatch()`, etc).

Added: `rst.ActiveConnection` property now returns the wrapper instance of the respective Connection object instead of connection string value.

Added: Parameters processing in recordset and connection `Open()` method.

Added: Support for `Connection.Properties` collection and `Recordset.Properties` collection.

Bug fix: Properties and methods that are supposed to return something meaningful incorrectly returned Lingo value instead of `VOID` in case of a error with failed flag set. Fixed

Bug fix: Several fixes of error checking code that incorrectly does not generate an error in case of inability to properly type cast Lingo value into ADO value and vice versa.

Bug fix: Assignment new value to the field's value property (`field.value=newValue` or `rst.field[index].value=newValue`) did not generate an error in case of inability to properly type cast Lingo value into ADO value. So, it silently did nothing in such cases. Assignment using two other methods (`rst.field[index]=newValue` or `rst.SetFieldValue(index,newValue)`) works properly. Fixed.

Bug fix: Xtra incorrectly allowed to create child xtra instances like `gADO=new(xtra "ADOxtra")`. Although everything works in this way, it is not supported. Use `gADO=xtra"ADOxtra"` instead. Fixed.

January 22nd, 2001 The first public release.

Technical details

Supported subset of ADO

ADO has rather large version history, since it is growing. ADO is COM-based, therefore it keeps backward compatibility while growing (in the most of cases). ADOxtra implements only the most common subset of ADO interfaces (defined since ADO version 2.0). Supporting interface version 2.0 allows ADOxtra to be used with almost all Windows systems with ADO preinstalled. All later ADO releases fully support interfaces of version 2.0. Furthermore, it is usually recommended to use later version of ADO, since Microsoft fixes some ADO bugs.

ADOxtra supports only four types of ADO objects: ADODB.Connection, ADODB.Recordset, ADODB.Field, ADODB.Property. These objects are usually enough for the most Director databasing applications. If you have to use functionality of later ADO versions or ADO extension components (ADOX) to use saved recordset, database management and other features, take a look at VbScriptXtra, which allows using ADO, ADOX, DAO via universal Automation technology right from Lingo. VbScriptXtra syntax differs slightly from ADOxtra syntax, but ADO is ADO, which is well documented at msdn.microsoft.com.

Default object's property

ADOxtra supports default object's property where appropriate. Usually default object's property is `Value`. For example, following lines make the same action:

```
val=rst.fields["FieldName"]
val=rst.fields["FieldName"].Value
```

Visual Basic uses different assignment operators for assigning reference to the object and assigning value of other data type. Different assignment operators allow VB interpreter to distinguish between using object reference and using the default property of that object. Lingo does not allow differentiating these two cases, therefore ADOxtra always uses default property in such cases (where Lingo allows). If you need to save object reference in a variable, use `ref` common property of ADOxtra wrapper object, as in a sample below:

```
fld=rst.fileds["FieldName"].ref
put fld.value
```

Default object's method (or indexed property)

Default object's method (indexed property) is not supported by ADOxtra, since Lingo syntax differences. In Visual Basic you can use:
`type=rst("FieldName").Type`
This sample will not work with ADOxtra. Instead use:
`type=rst.fields["FieldName"].Type`
Note square brackets [ ], which are used by ADOxtra instead of usual brackets ( ) in VB.

Events

The current version of ADOxtra does not support automation events.

Named method's arguments

Named arguments are not supported by ADOxtra since they are not supported by Lingo. In Visual Basic you may use following syntax:

```
obj.Method paramName:=actualValue
```

ADOxtra does not provide this feature.

Optional and missing method's arguments

Optional and missing arguments are supported by ADOxtra but Lingo requires you to use `VOID` value to indicate missing argument in the middle of the parameters list. Missing arguments in the end of the argument list may be safely skipped. Default values will be used by ADO object.

Passing parameters by reference

The current version of ADOxtra does not support parameters passed by reference, since Lingo does not support it for simple data types. There is a one case where this problem takes place with ADOxtra. It is an `Execute` method of a Connection object:

```
cnn.Execute CommandText, RecordsAffected, Options
```

In VB `RecordsAffected` gets the number of records affected by the executed operation. ADOxtra does not currently provide this feature. If you need it see VbScriptXtra, which allows using ADO, ADOX, DAO via universal Automation technology right from Lingo.

Arrays as arguments an returned values

The current version of ADOxtra does not support arrays.

Several ADO methods may accept optional arrays or return arrays. See VbScriptXtra, which allows using ADO, ADOX, and DAO via universal Automation technology right from Lingo.

Xtra-level methods, provided by ADOxtra:

`Init(bDebugMode)` - Initializes ADOxtra and optionally sets debugging mode for newly created wrapper instances.

`CreateObject(symObjectId)` - Creates specified object and ADOxtra wrapper instance to allow Lingo access to the newly created object.

`Version()` - Returns the version of ADOxtra.

`About()` - Returns the about information of ADOxtra.

`DateTimeListToFloat(propList)` - Returns the float representation of the date/time specified by a property List argument.

`FloatToDateTimeList(fltDateTime)` - Returns the property list with date/time information converted from float representation of the date/time.

`BuildConnectionString()` - Invokes the dialog box for choosing data source and other properties. Returns the string with connection information that may be used as connectionString while opening Connection object or Recordset object.

`xtra "ADOxtra".About()`

Syntax
```
strAbout=About(xtra"ADOxtra")
or
strAbout=xtra("ADOxtra").About()
```

Returns `String`: with ADOxtra about information.

Description Returns the about information of ADOxtra.

`xtra "ADOxtra".BuildConnectionString()`

Syntax
```
strConnectionString=BuildConnectionString(xtra"ADOxtra")
or
strConnectionString=xtra("ADOxtra").BuildConnectionString()
```

Returns `String`: connection string with connection parameters specified.

Description Invokes the dialog box for choosing data source and other properties. Returns the string with connection information, that may be used as `connectionString` while opening Connection object or Recordset object

`xtra "ADOxtra".CreateObject(symObjectId)`

Syntax

```
obj=CreateObject(xtra"ADOxtra",symObjectId)
or
obj=xtra("ADOxtra").CreateObject(symObjectId)
```

Parameters `symObjectId` - a symbol specifying which object to create. It can be either #Connection or #Recordset.

Returns `Object`: if successfully, returns new ADOxtra wrapper instance for newly created object (either Connection or Recordset).

`String`: if failed, returns string with error description.

Description This method is an analogue to Visual Basic's CreateObject. It is used to create new ADO objects. Therefore, it is a main entry point while using ADOxtra.

Sample

```
cnn=xtra("ADOxtra").CreateObject(#Connection)
rst=xtra("ADOxtra").CreateObject(#Recordset)
```

`DateTimeListToFloat()`

Syntax

```
fltDate=DateTimeListToFloat(lstDate)
Or
fltDate=DateTimeListToFloat(lstDate)
```

Parameters `lstDate` - a property list containing date/time information to be converted in float representation. The property list may contain following

properties: [#year: 2001, #month: 2, #day: 23, #Hour: 19, #Minute: 42, #Second: 23].

Returns `Float` representation of the specified date/time.

Description Converts "human readable" date/time property list to float representation of date/time values which is used in databases. Use this function to assign new value to table fields of date/time type.

The dateTimeList parameter may contain any subset of properties. If empty property list is passed, the current date and time will be returned in float form. If you specify only time part of the list, the current date will be used by default. If time part is missed it is set to midnight by default.

`FloatToDateTimeList()`

Syntax

```
lstDate=FloatToDateTimeList(fltDate)
```
or
```
lstDate=FloatToDateTimeList(fltDate)
```

Parameters `fltDate` - a float number which represent date/time value.

Returns `PropertyList` value that represents specified by fltDate date/time in "human readable" form: [#year: 2001, #month: 2, #day: 23, #DayOfWeek: 5, #Hour: 19, #Minute: 42, #Second: 23].

The #DayOfWeek property of returned list contains the day of the week in the range Sunday = 0, Monday = 1, and so on.

`String`: if failed, returns string with error description, which indicates that conversion, cannot be performed. This occurs if you specify a date/time value out of the present range (big positive or big negative).

Description Converts float representation of date/time value into the "human readable" date/time property list. Use this function to convert values of table fields of date/time type.

`xtra "ADOxtra".Init(bDebugMode)`

Syntax

```
bSuccess=Init(xtra"ADOxtra",bDebugMode)
or
bSuccess=xtra("ADOxtra").Init(bDebugMode)
```

Parameters

`bDebugMode` - a Boolean value specifying whether to set debug mode for newly created wrapper instances. Note: the value of this parameter affects wrapper instances created by further calls to `CreateObject` method.

When debug mode is set, wrapper instance outputs all information about errors in Messages window.

Returns

`true` if successful, `false` otherwise.

Description

Initializes ADOxtra and optionally sets debugging mode for newly created wrapper instances.

Debugging mode is highly recommended while investigation what ADOxtra can do, since it will output all error description information right in Messages window. There is no need to set debug mode in release versions.

You may safely call `Init` several times to enable or disable debug mode default setting.

`xtra "ADOxtra".Version()`

Syntax

```
strVersion=Version(xtra"ADOxtra")
or
strVersion=xtra("ADOxtra").Version()
```

Returns

`String`: with ADOxtra version information in a form: "ADOxtra.1.11.001"

Description

Returns the version information of ADOxtra.

Properties and methods provided by ADOxtra wrapper object

Wrapper object is a key component of ADOxtra. It is used to pass your Lingo method and property calls onto wrapped ADO object and return appropriate results, using automatic type casting and error handling mechanism.

Error handling properties:

`obj.DebugMode` - gets or sets debugging mode of the wrapper instance.

`obj.Succeeded` - returns true if previous wrapper access to the automation object has succeeded.

`obj.Failed` - returns true if previous wrapper access to the automation object has failed.

`obj.LastError` - returns the last error description if any. Error description usually is provided by ADO Object or COM library.

Quick object information properties:

`obj.methods` - returns a string with ADO object's methods supported by the wrapper object.

`obj.props` - returns a string with ADO object's properties supported by the wrapper object.

`obj.DebugMode`

Syntax

```
bDebugMode=obj.DebugMode
obj.DebugMode=bDebugMode
```

Gets Boolean value, which indicates whether wrapper currently in debug mode.

Sets Boolean value. Use true to set debug mode. Use `false` to clear debug mode of the wrapper instance.

Description ADOxtra object wrapper supports special debugging mode. While the debug mode is set wrapper instance outputs any error messages directly to the Message window every time error happens.

By default, ADOxtra wrappers created by `CreateObject` call inherit the xtra's default value for debugMode. You may change this default with `Init` xtra-level method. Wrappers created by other wrappers inherits this

setting.

Note: While you are just investigating ADO capabilities it is better to set Debug Mode by default, to ensure you always know if something goes wrong. It might be important while using cascading properties, since every property returning ADO Object will be wrapped by a new instance of ADOxtra wrapper. So, you may skip useful error description.

`obj.Failed`

Syntax `bResult=obj.Failed`

Returns Boolean value indicating whether last attempt to access wrapped ADO object has failed.

Description Use `Failed` property to check whether error occurred. Be careful with cascading properties, since cascaded access usually implemented using temporary wrapper instances. So, if error is encountered deeper then at the first level, you may not get a possibility to know that, unless Lingo error will be produced. Use debug mode to track down such conditions. Note also, that misspelled properties and methods will produce corresponding Lingo error in most cases.

See Debugging for related information.

`obj.LastError`

Syntax `strErrorDescription=obj.LastError`

Returns String with last error's description or empty string if the last call was successful.

Description Use `LastError` property to get the description of error occurred. Use debug mode to automatically get error descriptions in Messages window.

See Debugging for related information.

Sample

```
if obj.Failed then
  put obj.LastError
end if
```

obj.Succeeded

Syntax

```
bResult=obj.Succeeded
```

Returns

Boolean value indicating whether last attempt to access wrapped automation object has completed successfully.

Description

Use `Succeeded` property to check whether last operation with automation object completed successfully. Be careful with cascading properties, since cascaded access usually implemented using temporary wrapper instances. So, if error is encountered deeper then at the first level, you may not get a possibility to know that, unless Lingo error will be produced. Use debug mode to track down such conditions. Note also, that misspelled properties and methods will produce corresponding Lingo error in most cases.

See Debugging for related information.

obj.Props

Syntax

```
bResult=obj.Props
```

Returns

String: a RETURN separated list of properties supported by the wrapper for the given ADO object

Description

Returns the string with list of supported ADO properties one property per line. Use it to quickly display which properties you can use with the object.

Sample

```
rst=xtra("ADOxtra").CreateObject(#Recordset)
put rst.props
```

obj.Methods

Syntax `bResult=obj.Methods`

Returns String: a RETURN separated list of methods supported by the wrapper for the given ADO object

Description Returns the string with list of supported ADO methods one method per line. Use it to quickly display which methods you can use with the object.

Sample

```
cnn=xtra("ADOxtra").CreateObject(#Connection)
put cnn.methods
```

Properties and methods provided by ADOxtra wrapper object for ADODB.Connection

A Connection object represents a unique session with a data source.

Configure the connection before opening it with the `ConnectionString`, `ConnectionTimeout`, and `Mode` properties.

Set the `CursorLocation` property to client to invoke the Cursor Service for OLE DB, which supports batch updates.

Set the default database for the connection with the `DefaultDatabase` property.

Set the level of isolation for the transactions opened on the connection with the `IsolationLevel` property.

Specify an OLE DB provider with the `Provider` property.

Check provider settings with `Properties` collection.

Establish, and later break the physical connection to the data source with the `Open` and `Close` methods.

Execute a command on the connection with the `Execute` method.

Cancel asynchronous operation with `Cancel` method.

Manage transactions on the open connection, including nested transactions if the provider supports them, with the `BeginTrans`, `CommitTrans`, and `RollbackTrans` methods and the `Attributes` property.

Check the current connection state with the `State` property.

Examine errors returned from the data source with the `Errors` collection.

Read the version from the ADO implementation used with the `Version` property.

Obtain schema information about your database with the `OpenSchema` method.

`cnn.BeginTrans()`

Syntax `nLevel=cnn.BeginTrans()`

Returns Integer value indicating the nesting level of the transaction.

Description Calls the `BeginTrans()` method of the wrapped ADODB.Connection object. After you call the BeginTrans method, the provider will no longer instantaneously commit changes you make until you call `CommitTrans` or `RollbackTrans` to end the transaction.

Not supported in ADOxtraLite version.

`cnn.Cancel()`

Syntax `bSuccess=cnn.Cancel()`

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `Cancel()` method of the wrapped ADODB.Connection object. Use the `Cancel` method to terminate execution of an asynchronous method call (that is, a method invoked with the `adAsyncConnect`, `adAsyncExecute`, or `adAsyncFetch` option).

`cnn.Close()`

Syntax `bSuccess=cnn.Close()`

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `Close()` method of the wrapped ADODB.Connection object. Use the `Close` method to close a Connection to free any associated system resources. Closing an object does not remove it from memory; you can change its property settings and open it again later. To completely eliminate an object from memory, set the object variable to `VOID` after closing the object.

`cnn.CommitTrans()`

Syntax `bSuccess=cnn.CommitTrans()`

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `CommitTrans()` method of the wrapped ADODB.Connection object. Calling the `CommitTrans` method saves changes made within an open transaction on the connection and ends the transaction. Calling this method when there is no open transaction generates an error.

Not supported in ADOxtraLite version.

`cnn.Execute()`

Syntax `rst=cnn.Execute(commandText,recordsAffected,options)`

Parameters `commandText`

Optional. String value that contains the SQL statement, table name, stored procedure, or provider-specific text to execute.

`recordsAffected`

VOID. Does not supported in the current version. Set it to `VOID` if you have to specify options parameter.

`options`

Optional. Integer value that indicates how the provider should evaluate the `commandText` argument. It may be a bitmask of one or more `CommandTypeEnum` or `ExecuteOptionEnum` values.

Returns Newly created Recordset wrapper object that contains returned records if any.

Description Calls the `Execute()` method of the wrapped ADODB.Connection object. Using the Execute method on a Connection object executes whatever query you pass to the method in the `commandText` argument on the specified connection. If the `commandText` argument specifies a row-returning query, any results that the execution generates are stored in a new Recordset object. If the command is not a row-returning query, the

provider returns a closed Recordset object.

The returned Recordset object is always a read-only, forward-only cursor. If you need a Recordset object with more functionality, first create a Recordset object with the desired property settings, then use the Recordset object's Open method to execute the query and return the desired cursor type.

The contents of the `commandText` argument are specific to the provider and can be standard SQL syntax or any special command format that the provider supports.

`cnn.Open()`

Syntax

```
bSuccess=cnn.Open(connectionString,userID,password,options)
```

Parameters

`connectionString`

Optional. String value that contains connection information. See the `cnn.ConnectionString` property for details on valid settings.

`userID`

Optional. String value that contains a user name to use when establishing the connection.

`password`

Optional. String value that contains a password to use when establishing the connection.

`options`

Optional. Integer value that determines how the connection will be established, whether this method should return after (synchronously) or before (asynchronously) the connection is established. It may be one of the `ConnectOptionEnum` constants.

Returns

True (1) if function completed successfully, false (0) otherwise.

Description

Calls the `Open()` method of the wrapped ADODB.Connection object. Using the `Open` method on a Connection object establishes the physical

connection to a data source. After this method successfully completes, the connection is live and you can issue commands against it and process the results.

Use `cnn.adAsyncConnect` option to connect asynchronously. It is useful when connecting over busy LAN to MS SQL server. In this case check `cnn.state` property to determine whether connection succeeded.

`cnn.OpenSchema()`

Syntax `rstSchema=cnn.OpenSchema(queryType)`

Parameters `queryType`

Integer value that represents the type of schema query to run. It can be one of the `SchemaEnum` values.

Returns `Object`

new Recordset wrapper object that contains returned records.

Description Calls the `OpenSchema()` method of the wrapped ADODB.Connection object. The `OpenSchema` method returns self-descriptive information about the data source, such as what tables are in the data source, the columns in the tables, and the data types supported.

`cnn.RollbackTrans()`

Syntax `bSuccess=cnn.RollbackTrans()`

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `RollbackTrans()` method of the wrapped ADODB.Connection object. Calling the `RollbackTrans` method reverses any changes made within an open transaction and ends the transaction. Calling this method when there is no open transaction

generates an error.

Not supported in ADOxtraLite version.

`cnn.Attributes`

Syntax

```
put cnn.Attributes

cnn.Attributes=cnn.adXactAbortRetaining
```

Sets or gets

`Integer`

value that indicates transactions behavior. It may be 0 (default) or sum of one or more of the `XactAttributeEnum` values:

Description

Sets or gets the `Attributes` property of the wrapped ADODB.Connection object. Use the `Attributes` property to set or return characteristics of Connection objects.

When you set multiple attributes, you can sum the appropriate constants. If you set the property value to a sum including incompatible constants, an error occurs.

`cnn.ConnectionString`

Syntax

```
put cnn.ConnectionString
cnn.ConnectionString=strCnn
```

Sets or gets

`String` value with the information used to establish a connection to a data source.

Description

Sets or gets the ConnectionString property of the wrapped ADODB.Connection object. Indicates the information used to establish a connection to a data source. The `ConnectionString` property is read/write when the connection is closed and read-only when it is open.

Connection string is the string in form "`ArgumentName=ArgumentValue; OtherArgumentName=OtherValue`" that specify information

necessary to connect to the database. Use `BuildConnectionString` xtra level method to invoke a dialog for building connection string.

Duplicates of an argument in the `ConnectionString` property are ignored. The last instance of any argument is used.

Samples

MS Access databases

```
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=D:\TempDB.mdb; Mode=ReadWrite;"
```

MS Access password protected databases

```
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=D:\TempDB.mdb; Mode=ReadWrite; Jet
OLEDB:Database Password=PasswordHere"
```

MS Access databases via ODBC driver (DSNless connection):

```
connectionString="DRIVER={Microsoft Access Driver
(*.mdb)}; DBQ=D:\TempDB.mdb"
```

MS SQL Server:

```
connectionString="Provider=SQLOLEDB.1; Integrated
Security=SSPI; Persist Security Info=False; Initial
Catalog=DemoDB; Data Source=SqlServerName"
```

Oracle databases:

```
connectionString="Provider=MSDAORA.1; Password=psw;
User ID=admin; Data Source=srv; Persist Security
Info=True"
```

Remote **database connection:**

```
connectionString="Provider=MS Remote; Data
Source=YourDataSourceName; Remote
Server=http://YourWebServer; Internet Timeout=300000"
```

`cnn.ConnectionTimeout`

Syntax

```
put cnn.ConnectionTimeout

cnn.ConnectionTimeout=30
```

Sets or gets `Integer`

value that indicates, in seconds, how long to wait for the connection to open

Description Sets or gets the `ConnectionTimeout` property of the wrapped ADODB.Connection object. Sets or returns a value that indicates, in seconds, how long to wait for the connection to open. The `ConnectionTimeout` property is read/write when the connection is closed and read-only when it is open.

`cnn.CursorLocation`

Syntax

```
put cnn.CursorLocation

cnn.CursorLocation=cnn.adUseServer
```

Sets or gets `Integer`

value that can be set to one of the `CursorLocationEnum` values.

Description Sets or gets the `CursorLocation` property of the wrapped ADODB.Connection object. This property allows you to choose between various cursor libraries accessible to the provider. Usually, you can choose between using a client-side cursor library or one that is located on the server.

This property setting affects connections established only after the property has been set. Changing the `CursorLocation` property has no effect on existing connections.

`cnn.DefaultDatabase`

Syntax

```
put cnn.DefaultDatabase

cnn.DefaultDatabase="Pubs"
```

Sets or gets `String`

value that indicates the name of a database available from the provider.

Description Sets or gets the `DefaultDatabase` property of the wrapped ADODB.Connection object. Use the `DefaultDatabase` property to set or return the name of the default database on a specific Connection object.

If there is a default database, SQL strings may use an unqualified syntax to access objects in that database. To access objects in a database other than the one specified in the `DefaultDatabase` property, you must qualify object names with the desired database name. Upon connection, the provider will write default database information to the `DefaultDatabase` property.

Some providers allow only one database per connection, in which case you cannot change the `DefaultDatabase` property. Some data sources and providers may not support this feature, and may return an error or an empty string

`cnn.Errors`

Syntax

```
strProviderErrors=cnn.Error

strProviderError=cnn.Errors[i]

nProviderErrorsCount=cnn.Errors.count
```

Gets `String`

with provider error description.

Description Allows access to the Errors collection of the wrapped ADODB.Connection object. Any operation involving ADO objects can generate one or more provider errors. As each error occurs, one or more Error objects can be placed in the Errors collection of the Connection object. When another ADO operation generates an error, the Errors collection is cleared, and the new set of Error objects can be placed in the Errors collection.

cnn.IsolationLevel

Syntax

```
put cnn.IsolationLevel

cnn.IsolationLevel=cnn.adXactUnspecified
```

Sets or gets

`Integer`

value that indicates the isolation level of a Connection object. It can be set to one of the `IsolationLevelEnum` values

Description

Sets or gets the `IsolationLevel` property of the wrapped ADODB.Connection object. The setting does not take effect until the next time you call the `BeginTrans` method. If the level of isolation you request is unavailable, the provider may return the next greater level of isolation.

cnn.Mode

Syntax

```
put cnn.Mode

cnn.Mode=cnn.adModeReadWrite
```

Sets or gets

`Integer`

value that indicates the access permissions in use by the provider on the current connection. It can be set to one or more of the `ConnectModeEnum` values

Description

Sets or gets the `Mode` property of the wrapped ADODB.Connection object. Use the `Mode` property to set or return the access permissions in use by the provider on the current connection. You can set the `Mode` property only when the Connection object is closed.

When used on a client-side Connection object, the `Mode` property can only be set to `adModeUnknown`.

cnn.Properties

Syntax

```
nPropertiesCount=cnn.Properties.count

put cnn.Properties[propertyIndex]

cnn.Properties[propertyIndex]=newValue

put cnn.Properties[propertyIndex].Type

objProperty=cnn.Properties[propertyIndex].ref
```

Parameters

`propertyIndex`

String property name or Integer zero based index of the property object in `Connection.Properties` collection.

Sets or gets

`Any`

connection property value, if used in simple syntax.

`Object`

property object wrapper, if further cascading property is requested.

Description

Allows access to the `Value` property of the property objects or to the Property object itself in Properties collection of the wrapped ADODB.Connection object. Each Property object corresponds to a characteristic of the ADODB.Connection object specific to the provider.

Some properties are read only, others are read and write capable.

Sample

A sample script, which displays all properties of the passed connection object:

```
on ShowProperties cnn
  repeat with i= 0 to cnn.Properties.Count - 1
    put cnn.Properties[i].Name & "=" &
cnn.Properties[i]
  end repeat
end
```

`cnn.Provider`

Syntax `cnn.Provider`

`cnn.Provider="Microsoft.Jet.OLEDB.4.0"`

Sets or gets `String`

value that indicates the provider name.

Description Sets or gets the `Provider` property of the wrapped ADODB.Connection object. Use the `Provider` property to set or return the name of the provider for a connection. The `Provider` property is read/write when the connection is closed and read-only when it is open.

`cnn.State`

Syntax `put cnn.State`

Gets `Integer`

The state of the connection object. It can be a bitmask of the `ObjectStateEnum` values.

Description Returns the `State` property of the wrapped ADODB.Connection object. Indicates whether the state of the object is open or closed and asynchronous state as well.

`cnn.Version`

Syntax `put cnn.Version`

Gets `String`

The Version string.

Description Returns the `Version` property of the wrapped ADODB.Connection object. Use the Version property to return the version number of the ADO implementation.

Properties and methods provided by ADOxtra wrapper object for ADODB.Recordset

Recordset object represents the entire set of records from a base table or the results of an executed command. At any time, the Recordset object refers to only a single record within the set as the current record.

Recordset object consists of records (rows) and fields (columns).

There are four different recordset types defined in ADO:

Dynamic cursor — allows you to view additions, changes, and deletions by other users; allows all types of movement through the Recordset that doesn't rely on bookmarks; and allows bookmarks if the provider supports them.

Keyset cursor — behaves like a dynamic cursor, except that it prevents you from seeing records that other users add, and prevents access to records that other users delete. Data changes by other users will still be visible. It always supports bookmarks and therefore allows all types of movement through the Recordset.

Static cursor — provides a static copy of a set of records for you to use to find data or generate reports; always allows bookmarks and therefore allows all types of movement through the Recordset. Additions, changes, or deletions by other users will not be visible. This is the only type of cursor allowed when you open a client-side Recordset object.

Forward-only cursor — allows you to only scroll forward through the Recordset. Additions, changes, or deletions by other users will not be visible. This improves performance in situations where you need to make only a single pass through a Recordset.

Set the `CursorType` property prior to opening the Recordset to choose the cursor type, or pass a `CursorType` argument with the `Open` method. Some providers don't support all cursor types. If you don't specify a cursor type, ADO opens a forward-only cursor by default.

When you open a Recordset, the current record is positioned to the first record (if any) and the `BOF` and `EOF` properties are set to False. If there are no records, the `BOF` and `EOF` property settings are True.

You can use the `MoveFirst`, `MoveLast`, `MoveNext`, and `MovePrevious` methods and the `AbsolutePosition`, `AbsolutePage`, and `Filter` properties to reposition the current record, assuming the provider supports the relevant functionality. Forward-only Recordset objects support only the `MoveNext` method. When you use the `Move` methods to visit each record, you can use the `BOF` and `EOF` properties to determine if you've moved beyond the beginning or end of the Recordset.

Recordset objects can support two types of updating immediate and batch. In immediate updating, all changes to data are written immediately to the underlying data source once

you call the `Update` method. If a provider supports batch updating, you can have the provider cache changes to more than one record and then transmit them in a single call to the database with the `UpdateBatch` method. This applies to changes made with the `AddNew`, `Update`, and `Delete` methods. After you call the `UpdateBatch` method, you can use the `Status` property to check for any data conflicts in order to resolve them.

Use `Fields` collection to access fields of the current record and actually a recordset data. Also ADOxtra provides additional methods for reading and writing field data: GetFieldValue and SetFieldValue methods.

`rst.AddNew()`

Syntax `bSuccess=rst.AddNew()`

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `AddNew()` method of the wrapped ADODB.Recordset object. After you call the `AddNew` method, the new record becomes the current record and remains current after you call the Update method.

After you call the `AddNew` method, the new record becomes the current record and remains current after you call the `Update` method. If the Recordset object does not support bookmarks, you may not be able to access the new record once you move to another record. Depending on your cursor type, you may need to call the `Requery` method to make the new record accessible.

If you call `AddNew` while editing the current record or while adding a new record, ADO calls the Update method to save any changes and then creates the new record.

Not supported in ADOxtraLite version.

`rst.Cancel()`

Syntax `bSuccess=rst.Cancel()`

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `Cancel()` method of the wrapped ADODB.Recordset object. Use the Cancel method to terminate execution of an asynchronous `Open()` method call (that is, a method invoked with the `adAsyncExecute`, or `adAsyncFetch` option).

`rst.CancelBatch()`

Syntax `bSuccess=rst.CancelBatch(affectRecords)`

Parameters `affectRecords`

Optional. An Integer value that determines how many records the `CancelBatch` method will affect. The default value is `adAffectAll`. It can be one of `AffectEnum` values.

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `CancelBatch()` method of the wrapped ADODB.Recordset object. Use the `CancelBatch` method to cancel any pending updates in a Recordset in batch update mode. If the Recordset is in immediate update mode, calling `CancelBatch` without `adAffectCurrent` generates an error.

Not supported in ADOxtraLite version.

`rst.CancelUpdate()`

Syntax `bSuccess=rst.CancelUpdate()`

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `CancelUpdate()` method of the wrapped ADODB.Recordset object. Use the `CancelUpdate` method to cancel any changes made to the current row or to discard a newly added row. You cannot cancel changes to the current row or a new row after you call the `Update`

method.

Not supported in ADOxtraLite version.

`rst.Close()`

Syntax `bSuccess=rst.Close()`

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `Close()` method of the wrapped ADODB.Recordset object. Use the `Close` method to close a Recordset to free any associated system resources. Closing an object does not remove it from memory; you can change its property settings and open it again later. To completely eliminate an object from memory, set the object variable to VOID after closing the object.

If an edit is in progress while in immediate update mode, calling the `Close` method generates an error; instead, call the `Update` or `CancelUpdate` method first. If you close the Recordset object while in batch update mode, all changes since the last `UpdateBatch` call are lost.

`rst.Delete()`

Syntax `bSuccess=rst.Delete_(affectRecords)`

Note an underscore after the Delete word. Lingo processes delete keyword on its own, therefore the underscore is necessary to invoke Delete method of the wrapped object.

Parameters `affectRecords`

Optional. An Integer value that determines how many records the `CancelBatch` method will affect. The default value is `adAffectAll`. It can be one of `AffectEnum` values.

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `Delete()` method of the wrapped ADODB.Recordset object.

Using the `Delete` method marks the current record or a group of records in a Recordset object for deletion. If the Recordset object doesn't allow record deletion, an error occurs. If you are in immediate update mode, deletions occur in the database immediately. If a record cannot be successfully deleted (due to database integrity violations, for example), the record will remain in edit mode after the call to Update. This means that you must cancel the update with `CancelUpdate` before moving off the current record (for example, with `Close`, `Move`).

If you are in batch update mode, the records are marked for deletion from the cache and the actual deletion happens when you call the `UpdateBatch` method. (Use the `Filter` property to view the deleted records.)

Retrieving field values from the deleted record generates an error. After deleting the current record, the deleted record remains current until you move to a different record. Once you move away from the deleted record, it is no longer accessible.

If you nest deletions in a transaction, you can recover deleted records with the `RollbackTrans` method. If you are in batch update mode, you can cancel a pending deletion or group of pending deletions with the `CancelBatch` method.

Not supported in ADOxtraLite version.

`rst.GetFieldValue()`

Syntax `fieldValue=rst.GetFieldValue(fieldIndex)`

Parameters `fieldIndex`

Integer zero based index of the field object in `Recordset.Fields` collection

`fieldIndex`

String that identifies the field object in `Recordset.Fields` collection.

Returns `Any`

If operation competed successfully, this function returns the value of the field indicated by fieldIndex.

`Void`

If operation failed this function sets internal error flag and returns VOID

Description Gets the value of the recordset's field indicated by fieldIndex. It gets the `Recordset.Fields.Item(FieldIndex)`.Value of the wrapped ADODB.Recordset object. See ADOxtra typecasting topic for more details about how ADOxtra converts ADO value to Lingo value.

`rst.GetString()`

Syntax

```
strData=rst.GetString(stringFormat,numRows,columnDelimiter,rowDelimiter,nullExpr)
```

Parameters `stringFormat`

Integer value that specifies how recordset data should be converted to a string. It can be either VOID or `adClipString` (=2). In both cases method delimits rows by `RowDelimiter`, columns by `ColumnDelimiter`, and null values by `NullExpr`.

`numRows`

Optional. The number of rows to be converted in the Recordset. If `numRows` is not specified, or if it is greater than the total number of rows in the Recordset, then all the rows in the Recordset are converted.

`columnDelimiter`

String. Optional. A delimiter used between columns, if specified, otherwise the TAB character.

`rowDelimiter`

String. Optional. A delimiter used between rows, if specified, otherwise the RETURN character.

`nullExpr`

String. Optional. An expression used in place of a null value, if specified,

otherwise the empty string.

Returns `String`

If operation competed successfully, this function returns the string representation of recordset data.

Description Calls the `GetString()` method of the wrapped ADODB.Recordset object. Use it to get the string representation of recordset data.

`rst.MoveFirst()`

Syntax `bSuccess=rst.MoveFirst()`

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `MoveFirst()` method of the wrapped ADODB.Recordset object. Use the `MoveFirst` method to move the current record position to the first record in the Recordset.

`rst.MoveLast()`

Syntax `bSuccess=rst.MoveLast()`

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `MoveLast()` method of the wrapped ADODB.Recordset object. Use the `MoveLast` method to move the current record position to the last record in the Recordset. The Recordset object must support bookmarks or backward cursor movement; otherwise, the method call will generate an error.

`rst.MoveNext()`

Syntax `bSuccess=rst.MoveNext()`

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `MoveNext()` method of the wrapped ADODB.Recordset object. Use the `MoveNext` method to move the current record position one record forward (toward the bottom of the Recordset). If the last record is the current record and you call the `MoveNext` method, ADO sets the current record to the position after the last record in the Recordset (EOF is True). An attempt to move forward when the EOF property is already True generates an error.

`rst.MovePrevious()`

Syntax `bSuccess=rst.MovePrevious()`

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `MovePrevious()` method of the wrapped ADODB.Recordset object. Use the `MovePrevious` method to move the current record position one record backward (toward the top of the Recordset). The Recordset object must support bookmarks or backward cursor movement; otherwise, the method call will generate an error. If the first record is the current record and you call the `MovePrevious` method, ADO sets the current record to the position before the first record in the Recordset (`BOF` is True). An attempt to move backward when the `BOF` property is already True generates an error. If the Recordset object does not support either bookmarks or backward cursor movement, the `MovePrevious` method will generate an error.

`rst.Open()`

Parameters `source`

Optional. String value that contains an SQL statement, table name, or

stored procedure.

`activeConnection`

Optional. Either a ADODB.Connection object, or a String that contains ConnectionString parameters. Use `BuildConnectionString` xtra level method to invoke a dialog for building connection string.

`cursorType`

Optional. An integer value that determines the type of cursor that the provider should use when opening the Recordset. The default value is `adOpenForwardOnly`. It can be one of the `CursorTypeEnum` values.

`lockType`

Optional. An integer value that determines what type of locking (concurrency) the provider should use when opening the Recordset. The default value is `adLockReadOnly`. It can be one of the `LockTypeEnum` values.

`options`

Optional. Integer value that that indicates how the provider should evaluate the source argument. It may be a bitmask of one or more of the `CommandTypeEnum` or `ExecuteOptionEnum` values.

Syntax `bSuccess=rst.Open(source,activeConnection,cursorType,lockType,options)`

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `Open()` method of the wrapped ADODB.Recordset object. Using the Open method on a Recordset object opens a cursor that represents records from a base table or the results of a query.

`bSuccess=rst.Requery(options)`

Syntax `bSuccess=rst.Requery(options)`

Parameters `options`

Optional. Integer value that that indicates how the provider should evaluate the source argument. It may be a bitmask of one or more of the `CommandTypeEnum` or `ExecuteOptionEnum` values.

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `Requery()` method of the wrapped ADODB.Recordset object. Use the `Requery` method to refresh the entire contents of a Recordset object from the data source by reissuing the original command and retrieving the data a second time. Calling this method is equivalent to calling the `Close` and `Open` methods in succession. If you are editing the current record or adding a new record, an error occurs.

`rst.SetFieldValue()`

Syntax `bSuccess=rst.SetFieldValue(fieldIndex, newValue)`

Parameters `fieldIndex`

Integer zero based index or String that identifies the field object in `Recordset.Fields` collection.

`newValue`

New value to be placed into the `Value` property of the field object indicated by `fieldIndex`.

Returns True (1) if function completed successfully, false (0) otherwise.

Description Sets the value of the recordset's field indicated by `fieldIndex` or `fieldName`. It sets the `Recordset.Fields.Item(Index).Value` of the wrapped ADODB.Recordset object with `newValue`. See ADOxtra typecasting topic for more details about how ADOxtra converts Lingo value to ADO value.

Not supported in ADOxtraLite version.

`rst.Supports()`

Syntax `bIsSupported=rst.Supports(cursorOptions)`

Parameters `cursorOptions`

Integer value that consists of one or more of `CursorOptionEnum` values.

Returns `Boolean`

value that indicates whether all of the features identified by the `cursorOptions` argument are supported by the provider.

Description Calls the `Supports()` method of the wrapped ADODB.Recordset object. Use the Supports method to determine what types of functionality a Recordset object supports. If the Recordset object supports the features whose corresponding constants are in `cursorOptions`, the Supports method returns True. Otherwise, it returns False.

`rst.Update()`

Syntax `bSuccess=rst.Update()`

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `Update()` method of the wrapped ADODB.Recordset object. Use the Update method to save any changes you make to the current record of a Recordset object since calling the `AddNew` method or since changing any field values in an existing record. The Recordset object must support updates.

Not supported in ADOxtraLite version.

`rst.UpdateBatch()`

Syntax `bSuccess=rst.UpdateBatch(affectRecords)`

Parameters `affectRecords`

Optional. An Integer value that determines how many records the `CancelBatch` method will affect. The default value is `adAffectAll`. It can be one of `AffectEnum` values.

Returns True (1) if function completed successfully, false (0) otherwise.

Description Calls the `UpdateBatch()` method of the wrapped ADODB.Recordset object. Use the `UpdateBatch` method when modifying a Recordset object in batch update mode to transmit all changes made in a Recordset object to the underlying database.

If the Recordset object supports batch updating, you can cache multiple changes to one or more records locally until you call the `UpdateBatch` method. If you are editing the current record or adding a new record when you call the `UpdateBatch` method, ADO will automatically call the Update method to save any pending changes to the current record before transmitting the batched changes to the provider. You should use batch updating with either a keyset or static cursor only.

Not supported in ADOxtraLite version.

`rst.AbsolutePage`

Syntax

```
put rst.AbsolutePage

rst.AbsolutePage=pageIndex
```

Gets `Integer`

value that indicates on which page the current record resides or one of the `PositionEnum` values.

Sets `Integer`

value to move the current record to the first record of a particular page.

Description Sets or gets the `AbsolutePage` property of the wrapped ADODB.Recordset object. Use the `AbsolutePage` property to identify the page number on which the current record is located. Use the `PageSize` property to logically divide the Recordset object into a series of pages, each of which has the number of records equal to `PageSize` (except for the last page, which may have fewer records). The provider must support the appropriate functionality for this property to be available. Like the `AbsolutePosition` property, `AbsolutePage` is 1-based and equals 1 when the current record is the first record in the Recordset. Set this property to move to the first record of a particular page. Obtain the total number of pages from the `PageCount` property.

`rst.AbsolutePosition`

Syntax

```
put=rst.AbsolutePosition
```

```
rst.AbsolutePosition=recordIndex
```

Gets `Integer`

value that indicates the ordinal position of a Recordset object's current record or one of the `PositionEnum` values.

Sets `Integer`

value to move to a record based on its ordinal position in the Recordset object.

Description Sets or gets the `AbsolutePosition` property of the wrapped ADODB.Recordset object. Use the `AbsolutePosition` property to move to a record based on its ordinal position in the Recordset object, or to determine the ordinal position of the current record. The provider must support the appropriate functionality for this property to be available. Like the `AbsolutePage` property, `AbsolutePosition` is 1-based and equals 1 when the current record is the first record in the Recordset. You can obtain the total number of records in the Recordset object from the `RecordCount` property.

rst.ActiveConnection

Syntax

```
put rst.ActiveConnection

rst.ActiveConnection=objConnection

rst.ActiveConnection="Provider=
Microsoft.Jet.OLEDB.4.0; Data
Source=D:\Temp\TestDB.mdb;"
```

Gets

`Object`

ADOxtra wrapper for ADODB.Connection object that is the current Connection object of the recordset.

`Symbol #Null`

if there is no current connection which recordset is bound to.

Sets

`String`

contains a definition for a connection. In this case, the provider creates a new ADODB.Connection object using this definition and opens the connection. Use `BuildConnectionString` xtra level method to invoke a dialog for building connection string. See `cnn.ConnectionString` property for connection string samples.

`Object`

ADOxtra wrapper for opened ADODB.Connection object. Use `CreateObject(xtra"ADOxtra",#Connection)` to create such object.

`Symbol #Null`

To detach recordset from the connection use `#Null`. It is possible if `cursorLocation` is set to `adUseClient`.

Description

Sets or gets the `ActiveConnection` property of the wrapped ADODB.Recordset object. Use the `ActiveConnection` property to determine the Connection object over which the specified Recordset will be opened.

rst.BOF

Syntax `put rst.BOF`

Gets `Boolean`

value that indicates the current record position is before the first record in a Recordset object.

Description Returns the `BOF` property of the wrapped ADODB.Recordset object. 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 `BOF` property returns True (1) if the current record position is before the first record and False (0) if the current record position is on or after the first record. If either the `BOF` or `EOF` property is True, there is no current record.

rst.Bookmark

Syntax

```
bookmark=rst.Bookmark
rst.Bookmark=bookmark
```

Sets or gets `Float`

value that indicates a bookmark that uniquely identifies the current record in a Recordset object.

Description Sets or gets the `Bookmark` property of the wrapped ADODB.Recordset object. Use the `Bookmark` property to save the position of the current record and return to that record at any time. Bookmarks are available only in Recordset objects that support bookmark functionality.

rst.CacheSize

Syntax

```
put rst.CacheSize
rst.CacheSize=10
```

Sets or gets `Integer`

value that indicates the number of records from a Recordset object that are cached locally in memory.

Description Sets or gets the `CacheSize` property of the wrapped ADODB.Recordset object. Use the `CacheSize` property to control how many records the provider keeps in its buffer and how many to retrieve at one time into local memory.

`rst.CursorLocation`

Syntax

```
put rst.CursorLocation
rst.CursorLocation=rst.adUseServer
rst.CursorLocation=rst.adUseClient
```

Sets or gets `Integer`

value that can be set to one of the `CursorLocationEnum` values.

Description Sets or gets the `CursorLocation` property of the wrapped ADODB.Recordset object. This property allows you to choose between various cursor libraries accessible to the provider. Usually, you can choose between using a client-side cursor library or one that is located on the server. Recordset objects will automatically inherit this setting from their associated connections.

`rst.CursorType`

Syntax

```
put rst.CursorType
rst.CursorType=rst.adOpenForwardOnly
rst.CursorType=rst.adOpenDynamic
rst.CursorType=rst.adOpenKeyset
rst.CursorType=rst.adOpenStatic
```

Sets or gets `Integer`

value that determines the type of cursor that the provider should use when opening the Recordset. The default value is adOpenForwardOnly. It can be one of the CursorTypeEnum values.

Description Sets or gets the `CursorType` property of the wrapped ADODB.Recordset object. Use the `CursorType` property to specify the type of cursor that should be used when opening the Recordset object. ADOxtra provides following constants:

`rst.EditMode`

Syntax `put rst.EditMode`

Gets `Integer`

value that indicates the editing status of the current record. It can be one of the `EditModeEnum` values.

Description Returns the `EditMode` property of the wrapped ADODB.Recordset object. ADO maintains an editing buffer associated with the current record. This property indicates whether changes have been made to this buffer, or whether a new record has been created. Use the `EditMode` property to determine the editing status of the current record.

`rst.EOF`

Syntax `put rst.EOF`

Gets `Boolean`

value that indicates the current record position is after the last record in a Recordset object.

Description Returns the `EOF` property of the wrapped ADODB.Recordset object. 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 `BOF` property returns True (1) if the current record position is before the first record and False (0) if the current record position is on or after the first record. If either the `BOF` or `EOF` property is True, there is no current record.

`rst.Fields`

Syntax

```
oldValue=rst.Fields[fieldIndex]

fieldsCount=rst.Fields.Count

rst.Fields[fieldIndex]=newValue
```

Parameters fieldIndex

String field name or Integer zero based index of the field object in `Recordset.Fields` collection.

Sets or gets `Any`

recordset field value, if used in simple syntax.

`Object`

field object wrapper, if further cascading property is requested.

Description Allows access to the `Value` property of the field objects or to Field object itself in `Fields` collection of the wrapped ADODB.Recordset object. Each Field object corresponds to a column in the Recordset.

Sample

```
on ShowCurrentRecord rst
  repeat with i = 0 to rst.Fields.Count - 1
    put rst.Fields[i].Name & "=" & rst.Fields[i]
  end repeat
end
```

rst.Filter

Syntax

```
put rst.Filter

rst.Filter="LastName = 'Smith' AND FirstName = 'John'"
```

Sets or gets

`String`

Criteria string — a string made up of one or more individual clauses concatenated with AND or OR operators.

`Integer`

value that specifies the group of records to be filtered from a Recordset. It can be one of the `FilterGroupEnum` values.

Description

Sets or gets the `Filter` property of the wrapped ADODB.Recordset object. Use the `Filter` property to selectively screen out records in a Recordset object. The filtered Recordset becomes the current cursor.

rst.LockType

Syntax

```
put rst.LockType

rst.LockType=rst.adLockReadOnly

rst.LockType=rst.adLockPessimistic

rst.LockType=rst.adLockOptimistic

rst.LockType=rst.adLockBatchOptimistic
```

Sets or gets

`Integer`

value that determines what type of locking (concurrency) the provider should use when opening the Recordset. The default value is `adLockReadOnly`. It can be one of the `LockTypeEnum` values.

Description

Sets or gets the `LockType` property of the wrapped ADODB.Recordset object. Set the `LockType` property before opening a Recordset to specify what type of locking the provider should use when opening it. Read the property to return the type of locking in use on an open Recordset object.

The `LockType` property is read/write when the Recordset is closed and read-only when it is open.

`rst.MarshalOptions`

Syntax

```
put rst.MarshalOptions

rst.MarshalOptions=rst.adMarshalAll

rst.MarshalOptions=rst.adMarshalModifiedOnly
```

Sets or gets

`Integer`

value that can be set to one of the `MarshalOptionsEnum` values.

Description

Sets or gets the `MarshalOptions` property of the wrapped ADODB.Recordset object. When using a client-side Recordset, records that have been modified on the client are written back to the middle tier or Web server through a technique called marshaling, the process of packaging and sending interface method parameters across thread or process boundaries. Setting the `MarshalOptions` property can improve performance when modified remote data is marshaled for updating back to the middle tier or Web server.

`rst.MaxRecords`

Syntax

```
put rst.MaxRecords

rst.MaxRecords=10
```

Sets or gets

`Integer`

value that indicates the maximum number of records to return to a Recordset from a query.

Description

Sets or gets the `MaxRecords` property of the wrapped ADODB.Recordset object. Use the `MaxRecords` property to limit the number of records that the provider returns from the data source. The default setting of this property is zero, which means the provider returns all requested records.

The `MaxRecords` property is read/write when the Recordset is closed and read-only when it is open.

`rst.PageCount`

Syntax `put rst.PageCount`

Gets `Integer`

Indicates how many pages of data the Recordset object contains.

Description Returns the `PageCount` property of the wrapped ADODB.Recordset object. Use the `PageCount` property to determine how many pages of data are in the Recordset object. Pages are groups of records whose size equals the `PageSize` property setting. Even if the last page is incomplete because there are fewer records than the `PageSize` value, it counts as an additional page in the `PageCount` value. If the Recordset object does not support this property, the value will be -1 to indicate that the `PageCount` is indeterminable. See the `PageSize` and `AbsolutePage` properties for more on page functionality.

`rst.PageSize`

Syntax

```
put rst.PageSize

rst.PageSize=10
```

Sets or gets `Integer`

value that indicates how many records constitute one page in the Recordset.

Description Sets or gets the `PageSize` property of the wrapped ADODB.Recordset object. Use the `PageSize` property to determine how many records make up a logical page of data. Establishing a page size allows you to use the `AbsolutePage` property to move to the first record of a particular page.

This property can be set at any time, and its value will be used for

calculating the location of the first record of a particular page.

`rst.Properties`

Syntax

```
nPropertiesCount=rst.Properties.count
put rst.Properties[propertyIndex]
rst.Properties[propertyIndex]=newValue
put rst.Properties[propertyIndex].Type
objProperty=rst.Properties[propertyIndex].ref
```

Sets or gets

`Any`

recordset property value, if used in simple syntax.

`Object`

property object wrapper, if further cascading property is requested.

Description

Allows access to the `Value` property of the property objects or to the Property object itself in Properties collection of the wrapped ADODB.Recordset object. Each Property object corresponds to a characteristic of the ADODB.Recordset object specific to the provider.

Some properties are read only, others are read and write capable.

Sample

A sample script, which displays all properties of the passed recordset object:

```
on ShowProperties rst
  repeat with i= 0 to rst.Properties.Count - 1
    put rst.Properties[i].Name & "=" &
rst.Properties[i]
  end repeat
end
```

`rst.RecordCount`

Syntax

```
put rst.RecordCount
```

Gets `Integer`

Indicates the number of records in a Recordset object.

Description Returns the `RecordCount` property of the wrapped ADODB.Recordset object. Use the `RecordCount` property to find out how many records are in a Recordset object. The property returns -1 when ADO cannot determine the number of records or if the provider or cursor type does not support `RecordCount`. Reading the `RecordCount` property on a closed Recordset causes an error.

`rst.Sort`

Syntax

```
put rst.Sort

rst.Sort="lastName DESC, firstName ASC"
```

Sets or gets `String`

value that indicates one or more field names on which the Recordset is sorted, and whether each field is sorted in ascending or descending order.

Description Sets or gets the `Sort` property of the wrapped ADODB.Recordset object. This property requires the `CursorLocation` property to be set to `adUseClient`. A temporary index will be created for each field specified in the Sort property if an index does not already exist. The sort operation is efficient because data is not physically rearranged, but is simply accessed in the order specified by the index. Setting the Sort property to an empty string will reset the rows to their original order and delete temporary indexes.

Suppose a Recordset contains three fields named `firstName`, `middleInitial`, and `lastName`. Set the Sort property to the string, "`lastName DESC, firstName ASC`", which will order the Recordset by last name in descending order, then by first name in ascending order.

rst.Source

Syntax

```
put rst.Source

rst.Source="SELECT * FROM Authors"
```

Sets or gets

`String`

value that indicates the data source for a Recordset object.

Description

Sets or gets the `Source` property of the wrapped ADODB.Recordset object. Use the `Source` property to specify a data source for a Recordset object using one of the following: an SQL statement, a stored procedure, or a table name.

rst.State

Syntax

```
put rst.State
```

Gets

`Integer`

The state of the connection object. It can be a bitmask of the `ObjectStateEnum` values.

Description

Returns the `State` property of the wrapped ADODB.Recordset object. Indicates whether the state of the object is open or closed.

rst.Status

Syntax

```
put rst.Status
```

Gets

`Integer`

value that indicates the status of the current record with respect to batch updates or other bulk operations. It can be a sum of one or more of the RecordStatusEnum values.

Description Returns the `EditMode` property of the wrapped ADODB.Recordset object. Use the `Status` property to see what changes are pending for records modified during batch updating. You can also use the Status property to view the status of records that fail during bulk operations, such as when you call the `Resync`, `UpdateBatch`, or `CancelBatch` methods on a Recordset object. With this property, you can determine how a given record failed and resolve it accordingly.

Properties provided by ADOxtra wrapper object for ADODB.Field

Field object represents a column of data with a common data type.

Each Field object corresponds to a column in the Recordset. You use the `Value` property of Field objects to set or return data for the current record.

With Field object properties you can do the following:

Return the name of a field with the `Name` property.

View or change the data in the field with the `Value` property.

Return the basic characteristics of a field with the `Type`, `Precision`, and `NumericScale` properties.

Return the declared size of a field with the `DefinedSize` property.

Return the actual size of the data in a given field with the `ActualSize` property.

If the provider supports batch updates, you can resolve discrepancies in field values during batch updating with the `OriginalValue` and `UnderlyingValue` properties.

`fld.ActualSize`

Syntax `put fld.ActualSize`

Gets `Integer`

value that indicates the actual length of a field's value.

Description Returns the `ActualSize` property of the wrapped ADODB.Field object. Use the `ActualSize` property to return the actual length of a Field object's value. If ADO cannot determine the length of the Field object's value, the `ActualSize` property returns -1.

`fld.Attributes`

Syntax `put fld.Attributes`

Gets `Integer`

value that indicates characteristics of a Property object. It can be a bitmask of the `FieldAttributeEnum` values.

Description Returns the `Attributes` property of the wrapped ADODB.Field object. Use the `Attributes` property to return characteristics of Field object.

`fld.DefinedSize`

Syntax `put fld.DefinedSize`

Gets `Integer`

value that reflects the defined size of a field as a number of bytes.

Description Returns the `DefinedSize` property of the wrapped ADODB.Field object. Use the `DefinedSize` property to determine the data capacity of a Field object.

`fld.Name`

Syntax `put fld.Name`

Gets `String`

value that indicates the name of the Field object.

Description Returns the `Name` property of the wrapped ADODB.Field object. Use the `Name` property to retrieve the name of a field object.

`fld.NumericScale`

Syntax `put fld.NumericScale`

Gets `Integer`

value that indicates the number of decimal places to which numeric values will be resolved.

Description Returns the `NumericScale` property of the wrapped ADODB.Field object. Use the `NumericScale` property to determine how many digits to the right of the decimal point is used to represent values for a numeric Field object.

`fld.OriginalValue`

Syntax `put fld.OriginalValue`

Gets `Any`

`OriginalValue` property of the field object.

Description Returns the `OriginalValue` property of the wrapped ADODB.Field object. Use the `OriginalValue` property to return the original field value for a field from the current record.

`fld.Precision`

Syntax `put fld.Precision`

Gets `Integer`

value that indicates the maximum number of digits used to represent values.

Description Returns the `Precision` property of the wrapped ADODB.Field object. Use the `Precision` property to get the maximum number of digits used to represent values for a numeric Field object.

`fld.Type`

Syntax `put fld.Type`

Gets `Integer`

value that indicates the data type of a Field object. It may be one of the `DataTypeEnum` values.

Description Returns the `Type` property of the wrapped ADODB.Field object.

`fld.UnderlyingValue`

Syntax `put fld.UnderlyingValue`

Gets `Any`

`UnderlyingValue` property of the field object.

Description Returns the `UnderlyingValue` property of the wrapped ADODB.Field object. Use the `UnderlyingValue` property to return the current field value from the database. The field value in the `UnderlyingValue` property is the value that is visible to your transaction and may be the result of a recent update by another transaction. This may differ from the `OriginalValue` property, which reflects the value that was originally returned to the Recordset.

`fld.Value`

Syntax `val=fld.Value`

`fld.Value=newValue`

Sets or gets `Any`

`Value` property of the field object.

Description Sets or gets the `Value` property of the wrapped ADODB.Field object. Use the `Value` property to set or return data from Field objects.

Set operations are not supported in the ADOxtraLite version.

Properties provided by ADOxtra wrapper object for ADODB.Property

Property object represents a dynamic characteristic of an ADO object that is defined by the provider.

Dynamic properties are defined by the underlying data provider, and appear in the Properties collection for the appropriate ADO object. For example, a property specific to the provider may indicate if a Recordset object supports transactions or updating. These additional properties will appear as Property objects in that Recordset object's Properties collection.

A dynamic Property object has four built-in properties of its own:

The `Name` property is a string that identifies the property.

The `Type` property is an integer that specifies the property data type.

The `Value` property contains the property setting.

The `Attributes` property is an integer value that indicates characteristics of the property specific to the provider.

`prop.Attributes`

Syntax `put prop.Syntax`

Gets `Integer`

value that indicates characteristics of a Property object. It can be one or sum of the `PropertyAttributesEnum` values.

Description Returns the `Attributes` property of the wrapped ADODB.Property object.

`prop.Name`

Syntax `put prop.Name`

Gets `String`

value that indicates the name of the Property object.

Description Returns the `Name` property of the wrapped ADODB.Property object. Use the `Name` property to retrieve the name of a Property object.

`prop.Type`

Syntax `put prop.Type`

Gets `Integer`

value that indicates the data type of a Field object. It may be one of the `DataTypeEnum` values.

Description Returns the `Type` property of the wrapped ADODB.Property object.

`prop.Value`

Syntax `put prop.Value`

`prop.Value=newValue`

Sets or gets `Any`

Value property of the property object.

Description Sets or gets the `Value` property of the wrapped ADODB.Property object. Use the `Value` property to set or return data from Property objects.

ADO Enumerated Constants used by methods and properties provided by ADOxtra wrapper object

`AffectEnum` - Specifies which records are affected by an operation.

`CommandTypeEnum` - Specifies how a command argument should be interpreted.

`ConnectModeEnum` - Specifies the available permissions for modifying data in a Connection object.

`ConnectOptionEnum` - Specifies whether the `Open` method of a Connection object should return after (synchronously) or before (asynchronously) the connection is established.

`CursorLocationEnum` - Specifies the location of the cursor engine.

`CursorOptionEnum` - Specifies what functionality the `Supports` method should test for.

`CursorTypeEnum` - Specifies the type of cursor used in a Recordset object.

`DataTypeEnum` - Specifies the data type of a Field or Property.

`EditModeEnum` - Specifies the editing status of a record.

`ExecuteOptionEnum` - Specifies how a provider should execute a command.

`FieldAttributeEnum` - Specifies one or more attributes of a Field object.

`FilterGroupEnum` - Specifies the group of records to be filtered from a Recordset.

`IsolationLevelEnum` - Specifies the level of transaction isolation for a Connection object.

`LockTypeEnum` - Specifies the type of lock placed on records during editing.

`MarshalOptionsEnum` - Specifies, which records should be returned to the server.

`ObjectStateEnum` - Specifies whether an object is open or closed, connecting to a data source, executing a command, or fetching data.

`PositionEnum` - Specifies the current position of the record pointer within a Recordset.

`PropertyAttributesEnum` - Specifies the attributes of a Property object.

`RecordStatusEnum` - Specifies the status of a record with regard to batch updates and other bulk operations.

`SchemaEnum` - Specifies the type of schema Recordset that the `OpenSchema` method retrieves.

`XactAttributeEnum` - Specifies the transaction attributes of a Connection object.

ADO::AffectEnum

Specifies which records are affected by an operation.

Constant	Value	Description
`adAffectAll`	3	If there is not a Filter applied to the Recordset, affects all records. If the Filter property is set to a string criteria (such as "Author='Smith'"), then the operation affects visible records in the current chapter. If the `Filter` property is set to a member of the `FilterGroupEnum`, then the operation will affect all rows of the Recordset.
`adAffectAllChapters`	4	Affects all records in all sibling chapters of the Recordset, including those not visible via any `Filter` that is currently applied.
`adAffectCurrent`	1	Affects only the current record.
`adAffectGroup`	2	Affects only records that satisfy the current Filter property setting. You must set the `Filter` property to a `FilterGroupEnum` value.

ADO::CommandTypeEnum

Specifies how a command argument should be interpreted.

Constant	Value	Description
`adCmdUnspecified`	-1	Does not specify the command type argument.
`adCmdText`	1	Evaluates `commandText` parameter as a textual definition of a command or stored procedure call.
`adCmdTable`	2	Evaluates `commandText` parameter as a table name whose columns are all returned by an internally generated SQL query.

adCmdStoredProc	4	Evaluates commandText parameter as a stored procedure name.
adCmdUnknown	8	Default. Indicates that the type of command in the commandText parameter property is not known.

ADO::ConnectModeEnum

Specifies the available permissions for modifying data in a Connection object.

Constant	Value	Description
adModeRead	1	Indicates read-only permissions.
adModeReadWrite	3	Indicates read/write permissions.
adModeShareDenyNone	16	Allows others to open a connection with any permissions. Neither read nor write access can be denied to others.
adModeShareDenyRead	4	Prevents others from opening a connection with read permissions.
adModeShareDenyWrite	8	Prevents others from opening a connection with write permissions.
adModeShareExclusive	12	Prevents others from opening a connection.
adModeUnknown	0	Default. Indicates that the permissions have not yet been set or cannot be determined.
adModeWrite	2	Indicates write-only permissions.

ADO::ConnectOptionEnum

Specifies whether the Open method of a Connection object should return after (synchronously) or before (asynchronously) the connection is established.

Constant	Value	Description
cnn.adAsyncConnect	16	Opens the connection asynchronously.

`cnn.adConnectUnspecified`	-1	Default. Opens the connection synchronously.

ADO::CursorLocationEnum

Specifies the location of the cursor service.

Constant	Value	Description
adUseClient	3	Uses client-side cursors supplied by a local cursor library. Local cursor services often will allow many features that driver-supplied cursors may not, so using this setting may provide an advantage with respect to features that will be enabled. For backward compatibility, the synonym `adUseClientBatch` is also supported.
adUseServer	2	Default. Uses data-provider or driver-supplied cursors. These cursors are sometimes very flexible and allow for additional sensitivity to changes others make to the data source.

ADO::CursorOptionEnum

Specifies what functionality the Supports method should test for.

Constant	Value	Description
`adAddNew`	16778240	Supports the `AddNew` method to add new records.
`adApproxPosition`	16384	Supports the `AbsolutePosition` and `AbsolutePage` properties.
`adBookmark`	8192	Supports the `Bookmark` property to gain access to specific records.
`adDelete`	16779264	Supports the `Delete` method to delete records.
`adFind`	524288	Supports the `Find` method to locate a row in a Recordset.
`adHoldRecords`	256	Retrieves more records or changes the next position without committing all pending changes.

adIndex	1048576	Supports the Index property to name an index.
adMovePrevious	512	Supports the MoveFirst and MovePrevious methods, and Move or GetRows methods to move the current record position backward without requiring bookmarks.
adNotify	262144	Indicates that the underlying data provider supports notifications (which determines whether Recordset events are supported).
adResync	131072	Supports the Resync method to update the cursor with the data that is visible in the underlying database.
adSeek	2097152	Supports the Seek method to locate a row in a Recordset.
adUpdate	16809984	Supports the Update method to modify existing data.
adUpdateBatch	65536	Supports batch updating (UpdateBatch and CancelBatch methods) to transmit groups of changes to the provider.

ADO::CursorTypeEnum

Specifies the type of cursor used in a Recordset object.

Constant	Value	Description
adOpenDynamic	2	Uses a dynamic cursor. Additions, changes, and deletions by other users are visible, and all types of movement through the Recordset are allowed, except for bookmarks, if the provider doesn't support them.
adOpenForwardOnly	0	Default. Uses a forward-only cursor. Identical to a static cursor, except that you can only scroll forward through records. This improves performance when you need to make only one pass through a Recordset.
adOpenKeyset	1	Uses a keyset cursor. Like a dynamic cursor, except that you can't see records that other users

		add, although records that other users delete are inaccessible from your Recordset. Data changes by other users are still visible.
adOpenStatic	3	Uses a static cursor. A static copy of a set of records that you can use to find data or generate reports. Additions, changes, or deletions by other users are not visible.
adOpenUnspecified	-1	Does not specify the type of cursor.

ADO::DataTypeEnum

Specifies the data type of a Field, or Property.

Constant	Value	Description
adBoolean	11	Indicates a boolean value.
adBSTR	8	Indicates a null-terminated character string (Unicode).
adChar	129	Indicates a string value.
adCurrency	6	Indicates a currency value. Currency is a fixed-point number with four digits to the right of the decimal point. It is stored in an eight-byte signed integer scaled by 10,000.
adDate	7	Indicates a date value. A date is stored as a double, the whole part of which is the number of days since December 30, 1899, and the fractional part of which is the fraction of a day.
adDecimal	14	Indicates an exact numeric value with a fixed precision and scale.
adDouble	5	Indicates a double-precision floating-point value.
adEmpty	0	Specifies no value.
adInteger	3	Indicates a four-byte signed integer.
adGUID	72	Indicates a globally unique identifier (GUID).
adLongVarWChar	203	Indicates a long null-terminated Unicode string value.

adNumeric	131	Indicates an exact numeric value with a fixed precision and scale.
adSingle	4	Indicates a single-precision floating-point value.
adSmallInt	2	Indicates a two-byte signed integer.
adTinyInt	16	Indicates a one-byte signed integer.
adUnsignedInt	19	Indicates a four-byte unsigned integer.
adUnsignedSmallInt	18	Indicates a two-byte unsigned integer.
adUnsignedTinyInt	17	Indicates a one-byte unsigned integer.
adUserDefined	132	Indicates a user-defined variable.
adVarBinary	204	Indicates a binary value.
adVarChar	200	Indicates a string value.
adVarNumeric	139	Indicates a numeric value.
adVarWChar	202	Indicates a null-terminated Unicode character string.
adWChar	130	Indicates a null-terminated Unicode character string.

ADO::EditModeEnum

Specifies the editing status of a record.

Constant	Value	Description
adEditNone	0	Indicates that no editing operation is in progress.
adEditInProgress	1	Indicates that data in the current record has been modified but not saved.
adEditAdd	2	Indicates that the AddNew method has been called, and the current record in the copy buffer is a new record that has not been saved in the database.
adEditDelete	4	Indicates that the current record has been deleted.

ADO::ExecuteOptionEnum

Specifies how a provider should execute a command.

Constant	Value	Description
adAsyncExecute	16	Indicates that the command should execute asynchronously.
adAsyncFetch	32	Indicates that the remaining rows after the initial quantity specified in the cacheSize property should be retrieved asynchronously.
adAsyncFetchNonBlocking	64	Indicates that the main thread never blocks while retrieving. If the requested row has not been retrieved, the current row automatically moves to the end of the file.
adExecuteNoRecords	128	Indicates that the command text is a command or stored procedure that does not return rows (for example, a command that only inserts data). If any rows are retrieved, they are discarded and not returned.
adOptionUnspecified	-1	Indicates that the command is unspecified.

ADO::FieldAttributeEnum

Specifies one or more attributes of a Field object.

Constant	Value	Description
adFldCacheDeferred	4096	Indicates that the provider caches field values and that subsequent reads are done from the cache.
adFldFixed	16	Indicates that the field contains fixed-length data.
adFldIsChapter	8192	Indicates that the field contains a chapter value, which specifies a specific child recordset related to this parent field. Typically chapter fields are used with data shaping or filters.

adFldIsCollection	262144	Indicates that the field specifies that the resource represented by the record is a collection of other resources, such as a folder, rather than a simple resource, such as a text file.
adFldIsDefaultStream	131072	Indicates that the field contains the default stream for the resource represented by the record. For example, the default stream can be the HTML content of a root folder on a Web site, which is automatically served when the root URL is specified.
adFldIsNullable	32	Indicates that the field accepts null values.
adFldIsRowURL	65536	Indicates that the field contains the URL that names the resource from the data store represented by the record.
adFldLong	128	Indicates that the field is a long binary field. Also indicates that you can use the AppendChunk and GetChunk methods.
adFldMayBeNull	64	Indicates that you can read null values from the field.
adFldMayDefer	2	Indicates that the field is deferred—that is, the field values are not retrieved from the data source with the whole record, but only when you explicitly access them.
adFldNegativeScale	16384	Indicates that the field represents a numeric value from a column that supports negative scale values. The scale is specified by the NumericScale property.
adFldRowID	256	Indicates that the field contains a persistent row identifier that cannot be written to and has no meaningful value except to identify the row (such as a record number, unique identifier, and so forth).
adFldRowVersion	512	Indicates that the field contains some kind of time or date stamp used to track updates.
adFldUnknownUpdatable	8	Indicates that the provider cannot determine if

		you can write to the field.
adFldUnspecified	-1	Indicates that the provider does not specify the field attributes.
adFldUpdatable	4	Indicates that you can write to the field.

ADO::FilterGroupEnum

Specifies the group of records to be filtered from a Recordset.

Constant	Value	Description
adFilterAffectedRecords	2	Filters for viewing only records affected by the last Delete, Resync, UpdateBatch, or CancelBatch call.
adFilterConflictingRecords	5	Filters for viewing the records that failed the last batch update.
adFilterFetchedRecords	3	Filters for viewing the records in the current cache—that is, the results of the last call to retrieve records from the database.
adFilterNone	0	Removes the current filter and restores all records for viewing.
adFilterPendingRecords	1	Filters for viewing only records that have changed but have not yet been sent to the server. Applicable only for batch update mode.

ADO::IsolationLevelEnum

Specifies the level of transaction isolation for a Connection object.

Constant	Value	Description
adXactUnspecified	-1	Indicates that the provider is using a different isolation level than specified, but that the level cannot be determined.

adXactChaos	16	Indicates that pending changes from more highly isolated transactions cannot be overwritten.
adXactBrowse	256	Indicates that from one transaction you can view uncommitted changes in other transactions.
adXactReadUncommitted	256	Same as adXactBrowse.
adXactCursorStability	4096	Indicates that from one transaction you can view changes in other transactions only after they have been committed.
adXactReadCommitted	4096	Same as adXactCursorStability.
adXactRepeatableRead	65536	Indicates that from one transaction you cannot see changes made in other transactions, but that requiring can retrieve new Recordset objects.
adXactIsolated	1048576	Indicates that transactions are conducted in isolation of other transactions.
adXactSerializable	1048576	Same as adXactIsolated.

ADO::LockTypeEnum

Specifies the type of lock placed on records during editing.

Constant	Value	Description
adLockBatchOptimistic	4	Indicates optimistic batch updates. Required for batch update mode.
adLockOptimistic	3	Indicates optimistic locking, record by record. The provider uses optimistic locking, locking records only when you call the Update method.
adLockPessimistic	2	Indicates pessimistic locking, record by record. The provider does what is necessary to ensure successful editing of the records, usually by locking records at the data source immediately after editing.

adLockReadOnly	1	Indicates read-only records. You cannot alter the data.
adLockUnspecified	-1	Does not specify a type of lock. For clones, the clone is created with the same lock type as the original.

ADO::MarshalOptionsEnum

Specifies which records should be returned to the server.

Constant	Value	Description
adMarshalAll	0	Default. Returns all rows to the server.
adMarshalModifiedOnly	1	Returns only modified rows to the server.

ADO::ObjectStateEnum

Specifies whether an object is open or closed, connecting to a data source, executing a command, or retrieving data.

Constant	Value	Description
cnn.adStateClosed	0	Indicates that the object is closed.
cnn.adStateOpen	1	Indicates that the object is open.
cnn.adStateConnecting	2	Indicates that the object is connecting.
cnn.adStateExecuting	4	Indicates that the object is executing a command.
cnn.adStateFetching	8	Indicates that the rows of the object are being retrieved.

ADO::PositionEnum

Specifies the current position of the record pointer within a Recordset.

Constant	Value	Description

adPosBOF	-2	Indicates that the current record pointer is at `BOF` (that is, the `BOF` property is `True`).
adPosEOF	-3	Indicates that the current record pointer is at `EOF` (that is, the `EOF` property is `True`).
adPosUnknown	-1	Indicates that the `Recordset` is empty, the current position is unknown, or the provider does not support the `AbsolutePage` or `AbsolutePosition` property.

ADO::PropertyAttributesEnum

Specifies the attributes of a Property object.

Constant	Value	Description
`adPropNotSupported`	0	Indicates that the property is not supported by the provider.
`adPropRequired`	1	Indicates that the user must specify a value for this property before the data source is initialized.
`adPropOptional`	2	Indicates that the user does not need to specify a value for this property before the data source is initialized.
`adPropRead`	512	Indicates that the user can read the property.
`adPropWrite`	1024	Indicates that the user can set the property.

ADO::RecordStatusEnum

Specifies the status of a record with regard to batch updates and other bulk operations.

Constant	Value	Description
`adRecCanceled`	256	Indicates that the record was not saved because the operation was canceled.
`adRecCantRelease`	1024	Indicates that the new record was not saved because the existing record was locked.
`adRecConcurrencyViolation`	2048	Indicates that the record was not saved

		because optimistic concurrency was in use.
adRecDBDeleted	262144	Indicates that the record has already been deleted from the data source.
adRecDeleted	4	Indicates that the record was deleted.
adRecIntegrityViolation	4096	Indicates that the record was not saved because the user violated integrity constraints.
adRecInvalid	16	Indicates that the record was not saved because its bookmark is invalid.
adRecMaxChangesExceeded	8192	Indicates that the record was not saved because there were too many pending changes.
adRecModified	2	Indicates that the record was modified.
adRecMultipleChanges	64	Indicates that the record was not saved because it would have affected multiple records.
adRecNew	1	Indicates that the record is new.
adRecObjectOpen	16384	Indicates that the record was not saved because of a conflict with an open storage object.
adRecOK	0	Indicates that the record was successfully updated.
adRecOutOfMemory	32768	Indicates that the record was not saved because the computer has run out of memory.
adRecPendingChanges	128	Indicates that the record was not saved because it refers to a pending insert.
adRecPermissionDenied	65536	Indicates that the record was not saved because the user has insufficient permissions.
adRecSchemaViolation	131072	Indicates that the record was not saved because it violates the structure of the underlying database.

adRecUnmodified	8	Indicates that the record was not modified.

ADO::SchemaEnum

Specifies the type of schema Recordset that the OpenSchema method retrieves.

Constant	Value	Description
adSchemaAsserts	0	Returns the assertions defined in the catalog that are owned by a given user.
adSchemaCatalogs	1	Returns the physical attributes associated with catalogs accessible from the DBMS.
adSchemaCharacterSets	2	Returns the character sets defined in the catalog that are accessible to a given user.
adSchemaCheckConstraints	5	Returns the check constraints defined in the catalog that are owned by a given user.
adSchemaCollations	3	Returns the character collations defined in the catalog that are accessible to a given user.
adSchemaColumnPrivileges	13	Returns the privileges on columns of tables defined in the catalog that are available to, or granted by, a given user.
adSchemaColumns	4	Returns the columns of tables (including views) defined in the catalog that are accessible to a given user.
adSchemaColumnsDomainUsage	11	Returns the columns defined in the catalog that are dependent on a domain defined in the catalog and owned by a given user.
adSchemaConstraintColumnUsage	6	Returns the columns used by referential constraints, unique constraints, check constraints, and assertions, defined in the catalog and owned by a given user.

adSchemaConstraintTableUsage	7	Returns the tables that are used by referential constraints, unique constraints, check constraints, and assertions defined in the catalog and owned by a given user.
adSchemaCubes	32	Returns information about the available cubes in a schema (or the catalog, if the provider does not support schemas).
adSchemaDBInfoKeywords	30	Returns a list of provider-specific keywords.
adSchemaDBInfoLiterals	31	Returns a list of provider-specific literals used in text commands.
adSchemaDimensions	33	Returns information about the dimensions in a given cube. It has one row for each dimension.
adSchemaForeignKeys	27	Returns the foreign key columns defined in the catalog by a given user.
adSchemaHierarchies	34	Returns information about the hierarchies available in a dimension.
adSchemaIndexes	12	Returns the indexes defined in the catalog that are owned by a given user.
adSchemaKeyColumnUsage	8	Returns the columns defined in the catalog that are constrained as keys by a given user.
adSchemaLevels	35	Returns information about the levels available in a dimension.
adSchemaMeasures	36	Returns information about the available measures.
adSchemaMembers	38	Returns information about the available members.
adSchemaPrimaryKeys	28	Returns the primary key columns defined in the catalog by a given

		user.
adSchemaProcedureColumns	29	Returns information about the columns of rowsets returned by procedures.
adSchemaProcedureParameters	26	Returns information about the parameters and return codes of procedures.
adSchemaProcedures	16	Returns the procedures defined in the catalog that are owned by a given user.
adSchemaProperties	37	Returns information about the available properties for each level of the dimension.
adSchemaProviderTypes	22	Returns the (base) data types supported by the data provider.
adSchemaReferentialConstraints	9	Returns the referential constraints defined in the catalog that are owned by a given user.
adSchemaSchemata	17	Returns the schemas (database objects) that are owned by a given user.
adSchemaSQLLanguages	18	Returns the conformance levels, options, and dialects supported by the SQL-implementation processing data defined in the catalog.
adSchemaStatistics	19	Returns the statistics defined in the catalog that are owned by a given user.
adSchemaTableConstraints	10	Returns the table constraints defined in the catalog that are owned by a given user.
adSchemaTablePrivileges	14	Returns the privileges on tables defined in the catalog that are available to, or granted by, a given user.

`adSchemaTables`	20	Returns the tables (including views) defined in the catalog that are accessible to a given user.
`adSchemaTranslations`	21	Returns the character translations defined in the catalog that are accessible to a given user.
`adSchemaTrustees`	39	Reserved for future use.
`adSchemaUsagePrivileges`	15	Returns the USAGE privileges on objects defined in the catalog that are available to, or granted by, a given user.
`adSchemaViewColumnUsage`	24	Returns the columns on which viewed tables, defined in the catalog and owned by a given user, are dependent.
`adSchemaViews`	23	Returns the views defined in the catalog that are accessible to a given user.
`adSchemaViewTableUsage`	25	Returns the tables on which viewed tables, defined in the catalog and owned by a given user, are dependent.

ADO::XactAttributeEnum

Specifies the transaction attributes of a Connection object.

Constant	Value	Description
cnn.adXactAbortRetaining	262144	Performs retaining aborts. That is, calling `RollbackTrans` automatically starts a new transaction. Not all providers support this.
cnn.adXactCommitRetaining	131072	Performs retaining commits. That is, calling `CommitTrans` automatically starts a new transaction. Not all providers support this.

ADOxtra object wrapper - automatic type casting

ADOxtra performs automatic type casting to correctly transfer data between Lingo and ADO object and vice versa. Typecasting operations are implicitly performed by ADOxtra while processing Lingo method arguments, returning values, and property values.

Automation and Visual Basic supports rather large amount of data types. Lingo has its own Director specific data types. So, ADOxtra may not find suitable conversion in all cases, although it provides conversion in the most cases. If ADOxtra does not know how to convert the value it will report an error.

See Mapping Lingo types to ADO types and Mapping ADO types to Lingo types topics below for further details.

Mapping Lingo types to ADO types

This conversion is taking place when ADOxtra wrapper object passes any arguments to the wrapped ADO object. This includes assigning property values of the wrapped ADO object.

The table below describes which Lingo types are recognized by ADOxtra wrapper and into which types they are converted.

Lingo type	**Lingo Value (if any)**	**Automation type**
`Symbol`	`#Null`	NULL
`Symbol`	other symbols	String
`Integer`		signed integer 4 bytes
`Float`		Float 8 bytes
`String`		String
`Date`		Float
`VOID`	`VOID`	Missing value
ADOxtra wrapper		ADO object reference
BinaryXtra wrapper		SafeArray (Vector) of Bytes

The Symbol type is native to Director, ADO knows nothing about it, Furthermore, the actual integer value of any Director symbol is guaranteed to be the same only during current Director session. So, the special symbol value #Null is treated as ADO Null value. It is often used in databases and in VB to mark empty references to objects. Other Symbol values are converted to the corresponding String. So value #SomeSymbol will be converted to string "SomeSymbol".

Special note about Date values. The Date/Time value in ADO is actually represented as a float number where the whole part is the number of days since the 1st of January, 1901, and the fraction part represents the time. Director uses its own Date/Time values. ADOxtra provides a conversion of these values into ADO date/time values, but due to a bug in Director 8, the conversion gets the wrong results if the date value is outside this range: date(1901,12,14) to date(2038,1,19). Conversion of date values inside this range works correctly. Director 7.02 and Director 8.5 works correctly over all time period available to ADO.

ADOxtra provides a couple of routines for conversion float date representation into "human readable" form and vice versa: `DateTimeListToFloat` and `FloatToDateTimeList`

VOID Lingo values are usually treated as missing argument, therefore it is converted into corresponding COM Automation value which indicates missing argument.

Other ADOxtra wrapper as an argument is converted to the corresponding wrapped automation object reference. So you can safely use ADOxtra wrappers with corresponding object where other object expect it.

ADOxtra can handle large binary data contained in BinaryXtra wrapper. If such wrapper is passed as an argument, its data is converted to the SafeArray of unsigned chars. For more information about BinaryXtra see its documentation at http://www.xtramania.com/Documentation/BinaryXtra/.

Mapping ADO types to Lingo types

This conversion is taking place when ADOxtra wrapper returns any value returned by the wrapped ADO object. This includes getting property values of the wrapped ADO object.

The table below describes which ADO types are recognized by ADOxtra wrapper and into which Lingo types they are converted.

Automation type	**Lingo type**	**Value (if any)**
EMPTY	`VOID`	VOID
NULL	`Symbol`	#Null
Integer (signed/unsigned), 1,2,4 bytes	`Integer`	
Float 4,8 bytes	`Float`	
Numeric	`Float`	
Date	`Float`	
String	`String`	

Boolean	`Integer`	1 or 0
Currency	`Float`	
GUID	`String`	
ADO object reference	ADOxtra wrapper	
SafeArray of Bytes (BLOB, OLE, Image)	BinaryXtra wrapper	

Empty COM Automation values are converted to the VOID Lingo values.

Null value, which is often used in databases and in VB to mark empty references to objects, is converted to Lingo symbol #Null.

Any integer values are converted to 4 byte signed Integer value native to Director.

Any float or numeric or currency values are converted to 8 byte float value native to Director.

Boolean value in COM Automation usually represented as -1 for true and 0 for false. It is converted to integer values 1 and 0 accordingly.

GUID values are converted in string representation and then are converted into Lingo string values.

Date values are converted to the corresponding float values. Advanced date/time convertion routines will be available soon. ADOxtra provides a couple of routines for convertion float date representation into "human readable" form and vice versa: `DateTimeListToFloat` and `FloatToDateTimeList`

SafeArrays of Bytes (BLOB, OLE or Image database fields) are converted to the BinaryXtra wrappers containing those binary data. This conversion works if BinaryXtra is detected, otherwise 'Cannot convert ADO value to Lingo value' error is reported. BinaryXtra is free. For more information about BinaryXtra see its documentation at http://www.xtramania.com/Documentation/BinaryXtra/.

ADOxtra Samples

OpenRecordset - sample handler demonstrates how to open recordset to read or write data to or from Access database.

Useful scripts - several handlers useful while working with ADOxtra.

GuestBook - sample shocked GuestBook like movie which uses remote database. Sources of this sample are available at Downloads section

Remote Databasing Demo - guided tour of using ADOxtra with remote databases. Sources of this movie are available at Downloads section

GuestBook sample for ADOxtra

The Shockwave sample below demonstrates the general ADOxtra capabilities while accessing to the remote database.

The source of this movie is available in Downloads section.

For a general usage information about remote databasing see Documentation.

Feel free to play with the sample to see how your changes are actually written into database remotely.

OpenRecordset sample script

See ADOxtra castlib in Downloads section for this handler and other usefull Lingo for ADOxtra.

```
-- Handler creates a new recordset object
-- connects it to the MS Access database dbPath
-- sets access rights to read or read/write depending on
bReadWrite parameter
-- executes sql query and returns resulting recordset object if
successfull or
-- string with error description otherwise
on OpenRecordset dbPath, sql, bReadWrite
  if voidP(bReadWrite) then bReadWrite=false

  if voidP(sql) then return "OpenRecordset: Required parameter is
missing: sql"
  if voidP(dbPath) then return "OpenRecordset: Required parameter
is missing: dbPath"

  -- Creating recordset object
  rst=createObject(xtra "ADOxtra",#recordset)
  if not objectP(rst) then return rst

  -- Building connection string
```

```
  cnnStr="Provider=Microsoft.Jet.OLEDB.4.0;" -- Microsoft Jet
provider for MS Access databases
  cnnStr=cnnStr&"Data Source="&dbPath&";"
  if bReadWrite then
    cnnStr=cnnStr&"Mode=Read|Write;"
  else
    cnnStr=cnnStr&"Mode=Read;"
  end if

  rst.ActiveConnection=cnnStr
  if rst.failed then return rst.lastError

  if bReadWrite then
    rst.lockType=rst.adLockPessimistic
    rst.CursorType=rst.adOpenKeyset
  else
    rst.lockType=rst.adLockReadOnly
    rst.CursorType=rst.adOpenStatic
  end if

  rst.Open(sql)
  if rst.failed then return rst.lastError

  return rst
end
```

RemoteDemo sample for ADOxtra

The Shockwave sample below briefly describes how to make remote databasing working.

The source of this movie is available in Downloads section.

For a general usage information about remote databasing see Documentation.

Several useful scripts for ADOxtra

See ADOxtra castlib in Downloads section for this handler and other useful Lingo for ADOxtra.

```
-- Handler outputs the current record of a recordset rst
on ShowRecord rst
  if rst.state=0 then
    put "Recordset is closed."
  else if rst.bof or rst.eof then
    put "Current record does not exist."
  else
    repeat with i=0 to rst.Fields.Count-1
      desc=""
      desc=desc&&GetTypeName(rst.Fields[i].type)
&&"("&rst.Fields[i].type&")"
      attributes=rst.Fields[i].Attributes
```

```
      if CheckBit(attributes,rst.adFldIsNullable) then
desc=desc&&"Nullable"
      if CheckBit(attributes,rst.adFldMayBeNull) then
desc=desc&&"MayBeNull"
      if CheckBit(attributes,rst.adFldRowID) then
desc=desc&&"RowId"
      if CheckBit(attributes,rst.adFldUpdatable) then
        desc=desc&&"Updatable"
      else
        desc=desc&&"Readonly"
      end if

      put rst.Fields[i].Name &&
"="&&QUOTE&rst.Fields[i]&QUOTE&desc
    end repeat
  end if
end
-- Handler outputs all dynamic properties of the recordset or
connection object
-- Note that list of dynamic properties may very depending on
whether object is opened or not
-- Also the most of properties are read only when object is opened
on ShowProperties ref
  repeat with i = 0 to ref.Properties.count - 1
    desc=""
    desc=desc&&GetTypeName(ref.Properties[i].type)
    attributes=ref.Properties[i].Attributes
    if CheckBit(attributes,ref.adPropRequired) then
desc=desc&&"Required"
    if CheckBit(attributes,ref.adPropOptional) then
desc=desc&&"Optional"
    if CheckBit(attributes,ref.adPropRead) then desc=desc&&"Read"
    if CheckBit(attributes,ref.adPropWrite) then
desc=desc&&"Write"

    put ref.Properties[i].Name & "="
&QUOTE&ref.Properties[i]&QUOTE&desc
  end repeat
end
-- Handler returns the string description of the type returned by
type property of ADO objects
on GetTypeName type
  case type of
    11: return "boolean"
    8,129,203,202,200,130: return "string"
    6: return "currency"
    7: return "date"
    14,131,139: return "numeric"
    5,4: return "float"
    0: return "empty"
    3,2,16,19,18,17: return "integer"
```

```
    72: return "GUID"
    132: return "user-defined"
    204: return "binary"
    otherwise: return "unknown:"&type
  end case
end
-- Handler returns true if all bits in bitMask are set in val
on CheckBit val, bitMask
  return BitAnd(val,bitMask)=bitMask
end
```