

## Contents

---

Contents .....	1
Overview.....	2
Formate external access functions .....	2
ADO and OLE DB.....	2
ADO providers.....	2
What can be accessed using ADO?.....	2
Configuring external database access .....	3
ADO connect strings.....	4
General.....	4
Connect strings.....	4
Identifying the provider/driver to be used.....	4
Sample ADO connect strings - general.....	8
Sample ADO connect strings – using native OLE DB providers .....	9
Active Directory.....	9
Microsoft Access databases .....	9
Microsoft Indexing Service.....	9
Microsoft SQL Server databases.....	9
Oracle databases.....	9
Sample ADO connect strings – using ODBC drivers with a DSN.....	10
Microsoft Access databases .....	10
Microsoft Excel Workbooks .....	10
Microsoft SQL Server databases.....	10
Oracle databases.....	10
An alternate syntax.....	10
Sample ADO connect strings – using ODBC drivers without a DSN.....	11
Microsoft Access databases .....	11
Microsoft Excel workbooks .....	11
Microsoft SQL Server databases.....	11
Microsoft Visual FoxPro (with a database container).....	12
Microsoft Visual FoxPro (without a database container).....	12
Oracle databases.....	12
Paradox databases .....	12
Text files .....	12
ADO queries .....	13
General.....	13
Microsoft Active Directory .....	13
Microsoft Excel workbooks .....	18
SQL databases.....	19

## Overview

---

### Introduction

At times, you may need to look up data from an external source, i.e. from somewhere other than an incoming spool file. For example, the spool file may contain a customer code, but not a fax number or email address for the customer. If you wanted to email or fax the output, you would need to somehow find out what the email address or fax number was for a given customer code. If this is available in a database, you could use Formate's external lookup functions to access that database and return the required information.

### Formate external access functions

Formate provides three functions to let you do this: DbLookup(), DbField() and DbFillArray().

DbLookup() is designed to return information from a single record (e.g. for one customer), and initially returns a single field (e.g. the customer's name). It can also be used to request additional information from the same record (e.g. the various fields containing the customer's address), and the DbField() function can then be called to retrieve each one of these additional fields in turn.

If you need to return information from more than one record at a time, the DbFillArray() function allows you to do that, by creating an array with enough columns to hold all of the fields you requested, and enough rows to hold each returned record in its own row. You can then use the array functions such as GetCell() to retrieve the data from the array, or ArrayFind() to search for particular contents in the returned data.

### ADO and OLE DB

The DbLookup(), DbField() and DbFillArray() functions use Microsoft ActiveX Data Objects (ADO) technology to access external data. ADO is similar in purpose to the earlier Open Database Connectivity (ODBC) technology, in that it provides a uniform way to access various types of data, but ADO is generally more flexible and faster in operation.

ADO provides a simple-to-use front end based on an underlying data access technology called OLE DB, which you may also see mentioned in other documentation about ADO. Formate does not deal directly with OLE DB, but calls that it makes to ADO will make use of OLE DB services internally.

### ADO providers

ADO is distributed in the form of "providers", each of which is designed to provide access to one specific type of database. For example, there is an ADO provider for Microsoft SQL Server databases, and another one for Oracle databases. All of these providers offer a standardised way to access the database, regardless of what the underlying database technology is.

This greatly simplifies the task of writing routines to access a database, because you can generally use very similar routines regardless of which type of database is being accessed. You do still need to understand the structure of the data so you know which tables and fields to retrieve, but the method of retrieval is essentially the same for most types of database, whereas it might have differed greatly if you were accessing the data directly rather than through an ADO provider.

You must have the appropriate provider for the data source you need to access. The Microsoft Data Access Components (MDAC) are supplied with Formate (in the \MDAC folder on the Formate CD) and include a small selection of providers such as MS-Access and SQL Server. We do not supply any other providers, so if you need one which isn't there you will have to obtain it yourself from the supplier.

Please note that if you're having trouble accessing your data source, Formate support staff may be able to provide advice but no warranty is made in this area because given the enormous number of systems you might want to access, they couldn't possibly be familiar with them all or with the many tasks you might be trying to accomplish.

### What can be accessed using ADO?

Any database can be accessed, so long as there is an ADO provider for it. This includes almost all databases in common use today. For example, MS SQL Server, Oracle, MS-Access, and so on.

Perhaps less obviously, there are also providers for many other non-database sources of information. For example, there is a provider for the filesystem, so you can use ADO to find out what files are in a given folder. There is also a provider for Active Directory, so you can find out what computers or users are present in your domain.

Formate can provide access to any source for which you have an ADO provider, because the external access routines simply let ADO do the work, so they don't need to have any pre-built knowledge of the type of database you are accessing – the ADO provider takes care of that.

### Configuring external database access

To access external data from your Form Type, you must first set up a Database Connect String through the “External Lookup” tab in the Form Type Properties window. This lets you specify what the “connect string” will be, which is a string of information that tells ADO which data source you wish to connect to. See “ADO Connect strings” for more information.

These connect strings could have been entered directly into the DbLookup() and DbFillArray() functions wherever they appear in a script, but instead the strings are defined on the External Lookup tab and the functions simply include the ID or Name of one of those strings so that Formate looks up the contents of the string for you.

This has several advantages, not least of which is that you may need to use the same connection at several different places in your scripts. You only need to set it up once on the External Lookup tab, then each function call in a script can refer to that connection. If you find it doesn't work properly, or if you need to switch over to another data source later, you can do so in one step, just by changing the connect string – you don't need to find all the DbLookup() and DbFillArray() calls and change them all individually.

Once you have created a connect string, you must then include one of the external access functions, DbLookup() or DbFillArray() in your script, specifying a query string. This query tells ADO what information you want to retrieve from the source, and can include details such as the names of the fields you want, plus any criteria you need to apply (e.g. returning data only from the record where the customer code is a certain number). The exact content of this string will vary somewhat according to the type of data source you are accessing, and it's beyond the scope of this document to provide detailed examples for every one of the hundreds of possible sources. Some examples are provided for common data sources in the “Queries” section listed below, but otherwise you will require some knowledge of how to work with the particular data source in order to make up your own queries.

## ADO connect strings

---

### General

This section gives some suggestions about how to create ADO connect strings and how to query information from data sources. It is not a comprehensive description of ADO as that would run to many hundreds of pages. Nor does it attempt to cover every possible data source, as there are hundreds of those too, and periodically they are updated or new ones appear.

Some of the more common sources are covered here by working samples that you can adapt to your own needs. If you have to work with any source not covered here you may be able to develop your connect strings and queries based on ones for a similar source which *is* covered here. If not, you will need to obtain more information about your specific data source, either from the web or by contacting the supplier or manufacturer of the data source. Formate support staff may be able to help if it happens to be a source they have come across before, but they cannot undertake to be familiar with every possible source so they may refer you to the supplier/manufacturer.

### Connect strings

An ADO connect string contains the information that Formate needs in order to locate and gain access to a data source (usually a particular database). This may include items such as the type of database (e.g. SQL Server, FoxPro, MS-Access, Oracle, etc) , the path to the database, and a user ID and password if the database is password-protected. It may also contain optional additional information about your preferences, for example how many records should be held in a memory cache in order to improve performance.

External lookups are carried out via the DbLookup() and DbFillArray() script commands, and each of these needs you to provide it with an ADO connect string so that it knows how to make the connection.

To decide what should go into your connect string, the easiest way is normally to use another application such as Visual Basic version 6 that allows you to graphically build a connection to the data source, then copy the connect string that the application created. Otherwise, consult the rest of this section plus the documentation for your data source or its ADO driver.

Items in a connect string are separated by a semi-colon, and the string may contain as many items as required. Items generally have a name and a value separated by an equals sign, e.g. "Driver={Sql Server}", though there are some items where the presence of the name alone is all that's required, so there will be no equals sign and no value.

Formate communicates with a data source through a "provider" or "driver", which provides access to one particular type of database. There are a few items in the connect string which are interpreted by ADO itself, such as the one which tells it which provider to use, and these are effectively standardised. The content of the remainder of the string is entirely up to each provider, so unfortunately there are many variations and they can't all be detailed here. Some of the more common items *are* covered here, but for any provider which isn't mentioned or if you have complex requirements, you may need to contact the supplier of that provider and request assistance if you can't find any more detailed documentation elsewhere on how to use it.

### Identifying the provider/driver to be used

Virtually all connect strings will contain an item which identifies the provider and/or driver to be used. There are several ways in which this can be done, however, depending on which provider/driver you need to use.

#### *Using native OLE DB providers*

ADO and the underlying OLE DB technology can use providers which are specifically written to support OLE DB (and therefore ADO). These are known as "native" OLE DB providers, and generally provide the best performance. They are typically accessed by including a "PROVIDER=" item in the string, e.g. "PROVIDER=SQLOLEDB" to use the native OLE DB provider for SQL Server.

You will then need to add the appropriate items for the type of provider you are using. This will almost always include a database name or location, plus user ID and password if access to the database is password-protected, but it may also include many other items. The sample connect strings later in this topic include some such possible items for various providers.

### *Using ODBC drivers*

If you don't have a native OLE DB provider for the database you need to access, but you do have an older ODBC driver, you can use that instead. There is a native OLE DB provider specifically for this purpose, called the "OLE DB Provider for ODBC". This accepts commands from ADO, converts them into ODBC format and passes them on to the ODBC driver. When the driver returns data, the provider converts it back into OLE DB format and passes it on to ADO. This naturally involves some extra time while converting the data back and forth, so it's better to use a native OLE DB provider if there is one, but at least you can use the older ODBC driver if that's all you have.

There are two ways to use an ODBC driver from ADO, and they are covered separately in the next two sections.

### *Using ODBC drivers through a DSN*

ODBC supports the concept of defining all the characteristics of a database connection in an entity known as a Data Source Name or DSN, which is simply a convenient way to store connection information that would otherwise have to be specified as a number of separate items in a connection string. It therefore makes your connection string very simple because in many cases all you need to do is provide the name of the DSN, but it has a few disadvantages.

Firstly, if you want to change the contents, rather than just changing the connection string within Formate you need to use the separate ODBC Data Source Administrator applet (usually available through the Administrative Tools folder in Control Panel). Also, if anyone else changes the contents of the DSN, your database connection may no longer work. Finally, if you want to use the Form Type on another machine you will have to recreate the DSN manually on that machine because it isn't part of Formate and won't be included in any export files you may create through Formate.

Nonetheless, if you already have a DSN which works with the database you want to use, it may make sense to use that DSN rather than try to set up a new DSN-less connection string (which is covered in the next section).

There are three types of DSN – User, System, and File (the ODBC Data Source Administrator applet allows you to create whichever type you want, and lists them on separate tabs). User DSNs are only available to the currently logged in user, so if you create one while logged in as one user and then have Formate running as a different user, the connection will fail. It's therefore normally safer to use System DSNs, which are available to all users.

The syntax for a User or System DSN is

```
"PROVIDER=MSDASQL;DSN=Name"
```

where "Name" is the name you gave the DSN when you created it.

Similarly, the syntax for a File DSN is:

```
"PROVIDER=MSDASQL;FILEDSN=Path"
```

where "Path" is the full path of the DSN file.

MSDASQL (the name of the OLE DB Provider for ODBC) is the default OLE DB provider so the "Provider=MSDASQL;" part isn't strictly necessary, but it makes your intention clear and may also prevent problems in future in the unlikely event that the default provider is changed for some reason.

If the DSN contains all the necessary information to make the connection, you won't need to provide any additional items in the connection string. In some cases, however, you may need to add items such as user ID, password and database path, either because they are not present in the DSN, or because you need to over-ride them to use a different value. The sample connect strings in later sections include some such possible details for various ODBC drivers.

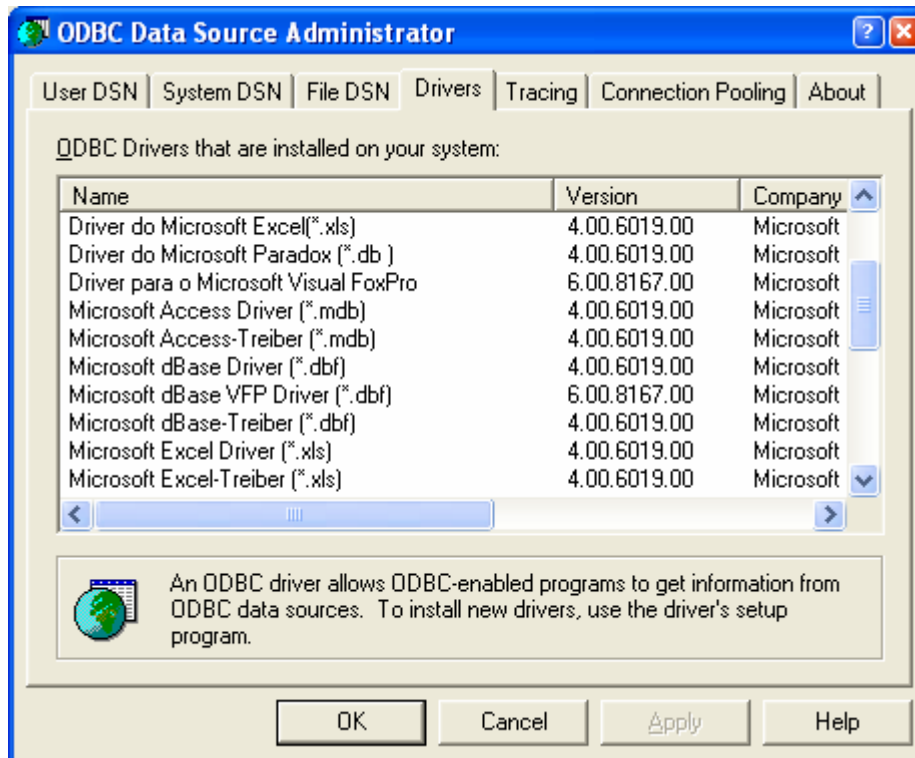
### Using ODBC drivers without a DSN

It's possible to set up a "DSN-less" ODBC connection, i.e. one which doesn't use a Data Source Name at all. This has the advantage that it saves the time which would have been spent configuring the DSN, and it makes Formate independent of any later changes that might have been made to the DSN which could have interfered with database access. It also means that if you copy the Form Type to another machine, all the necessary information to make the connection will go with it, which wouldn't be the case if a DSN was used because you would also need to manually set up an identical DSN on the new machine.

To set up a DSN-less ODBC connection, first add this item to tell ADO to use the OLE DB Provider for ODBC: "Provider=MSDASQL".

This connects you to the ODBC conversion routines but doesn't as yet tell ADO which ODBC driver to use. There is one of those for each different type of database, so you need to add a driver item to tell it which ODBC driver to use, e.g. "DRIVER={Microsoft Access Driver (\*.mdb)}".

The name of a driver can be found through the Drivers tab in the ODBC Data Source Administrator applet (usually available through the Administrative Tools folder in Control Panel). The name you use in the connect string should be letter-for-letter identical with the one shown on the "Name" column, including all spaces and punctuation, but with an opening and closing brace ("{" and "}") surrounding it:



An example of the complete definition of which provider/driver to use for an ODBC connection to an MS-Access database could therefore be:

```
"Provider=MSDASQL;DRIVER={Microsoft Access Driver (*.mdb)}".
```

Note the semi-colon separating the two items, and the opening and closing braces around the driver name.

If you don't see the ODBC driver you need in the list on the Drivers tab, it means the driver isn't installed on this machine so you need to obtain and install it.

MSDASQL (the OLE DB Provider for ODBC ) is the default OLE DB provider so the "Provider=MSDASQL;" part of the connection string isn't strictly necessary, but it makes your intention clear and may also prevent problems in future in the unlikely event that the default provider is changed for some reason.

## Trilogic White Paper - Accessing external data from Formate

---

Since this method does not refer to a DSN, the data which would have been used from the DSN must be included in the connect string instead, so you will normally need additional items for database name or path, user ID, password and so on. The sample connect strings in “Sample connect strings – using ODBC drivers without a DSN” include some such possible details for various ODBC drivers.

## Sample ADO connect strings - general

---

Where an item has a blank value (e.g. for a password where the current setting is blank), the equals sign should be immediately followed by a semi-colon, e.g. "PASSWORD=;". The semi-colon should still be used in this case even if the password is the last item in the string (apart from this particular situation, there is normally no need to add a semi-colon after the last item, although it does no harm).

In all these samples, where values such as database names, paths, user IDs, passwords etc are required, simple default values are used. Obviously, in order to use these strings you will need to replace these default values with the correct ones for your system.

Where some of the provider names include a version number (e.g. "Microsoft.Jet.OLEDB.3.51"), it isn't possible to keep this help up to date with new versions which may come out later, so if the given version doesn't work you may need to investigate whether you actually have a later version installed which needs a later version number (e.g. "Microsoft.Jet.OLEDB.4.0").

Where a string includes the "SERVER=" or "DATA SOURCE=" item, the value you supply should normally be the name of the server on which the database is to be found. Alternately, if TCP/IP is being used to connect to the database, it's often possible to supply the IP address rather than the server name.

The "DBQ=" or "DATABASE=" item normally needs either the name of the database (when using systems such as SQL Server which provide direct access regardless of where the data is physically held), or the path of the database file (when using simpler systems such as MS-Access).

Although quote marks are used to surround the strings here for illustrative purposes, you should not include those surrounding quote marks when entering a connect string in Formate.

A sample connect string may be split onto more than one line due to limitations of the available screen width in which to display it. Every connect string, however, is in reality always a 1-line string so you should not attempt to place linefeeds or carriage returns in the string when entering it into Formate.

---

## Sample ADO connect strings – using native OLE DB providers

### Active Directory

```
“Provider=ADsDSOObject;Encrypt Password=False;Data Source=Active Directory Provider;Mode=Read;Bind Flags=0;ADSI Flag=-2147483648”
```

This allows you to access the Active Directory database in which recent MS-Windows server operating systems can hold information about users, computers, domains, printers, shared drives, and various other resources.

The Active Directory Services Interface (ADSI) provider is supplied with Windows 2000 and Xp, but not with earlier versions of Windows. For Windows NT etc, see the following Microsoft web page which includes links to download the provider if you don't already have it:

[www.microsoft.com/ntserver/nts/downloads/other/ADSI25/default.asp](http://www.microsoft.com/ntserver/nts/downloads/other/ADSI25/default.asp)

### Microsoft Access databases

```
"PROVIDER=Microsoft.Jet.OLEDB.3.51;DATA SOURCE=C:\MyAccessDatabase.mdb;USER ID=jsmith;PASSWORD=secret;"
```

This should work if you have MS-Access 97. If you have MS-Access 2000 or later, you will most likely have a newer version of the Jet database engine provider, so you may need to replace the “3.51” with “4.0”.

Note that when using the native OLE DB provider for MS-Access, you cannot use a DSN in the connect string (i.e. “DSN=” is not allowed), because DSNs are used by ODBC drivers, not by OLE DB providers. Instead, you must supply the required information as separate items, as per the sample above where items are included to identify the database path and user login to be used.

In many cases, MS-Access databases will not be password-protected, in which case the user ID and password are ignored and generally need not be supplied.

The MS-Access provider is included in the Microsoft Data Access Components (MDAC) setups which are contained in the \MDAC folder on the Formate CD.

### Microsoft Indexing Service

```
“Provider=MSIDX.S.1;Data Source=c:\source.dat”
```

### Microsoft SQL Server databases

```
“PROVIDER=SQLOLEDB;DATA SOURCE=MySQLServer;DATABASE=Customers;USER ID=jsmith;PASSWORD=secret;”
```

Unusually, the SQL Server OLE DB provider alternately allows the use of a DSN if appropriate (assuming the DSN is set up to connect to a SQL Server database), e.g.

```
"PROVIDER=SQLOLEDB;DSN=MyServerDSN;DATABASE=Customers;UID=sa;PWD=secret;"
```

The SQL Server provider is included in the Microsoft Data Access Components (MDAC) setups which are contained in the \MDAC folder on the Formate CD.

### Oracle databases

```
"PROVIDER=MSDAORA;DATA SOURCE=MyOracleServer;USER ID=jsmith;PASSWORD=secret;"
```

---

## Sample ADO connect strings – using ODBC drivers with a DSN

### Microsoft Access databases

```
"PROVIDER=MSDASQL;DSN=MyAccessDSN;UID=jsmith;PWD=secret;"
```

### Microsoft Excel Workbooks

Note: If you have a license for the Formate Excel module, you can use functions such as `XIOpenWorkbook()` and `XIGetValues()` to read data from an Excel workbook, which is generally more reliable, more flexible, and simpler to set up than using external access via an ODBC driver.

```
"PROVIDER=MSDASQL;DSN=MyExcelDSN "
```

For additional information about accessing Excel workbooks, see the “Microsoft Excel Workbooks” section in the “Sample connect strings – using ODBC drivers without a DSN” topic below. Much of the information given there also relates to accessing Excel through a DSN.

### Microsoft SQL Server databases

```
"PROVIDER=MSDASQL;DSN=MySqlServer;DATABASE=Customers;UID=jsmith;PWD=secret;"
```

### Oracle databases

```
“PROVIDER=MSDASQL;DSN=MyOracleDSN;UID=jsmith;PWD=secret;”
```

### An alternate syntax

ADO provides an alternate way to specify the same information as shown in the previous examples in this section, the difference being that in the previous examples the information other than the provider and DSN names is passed on to the relevant ODBC driver for it to interpret, whereas in this alternate form, ADO itself interprets the items and then passes them on to the relevant driver in the format it requires. This means that this alternate syntax remains consistent regardless of which ODBC driver you are using. For example, you could use an MS-Access DSN as follows:

```
“PROVIDER=MSDASQL;DATA SOURCE=MyAccessDSN;USER ID=jsmith;PASSWORD=secret;”
```

"DATA SOURCE" is equivalent to "DSN" in the previous examples, "USER ID" to "UID", and "PASSWORD" to "PWD". If the underlying provider needs to identify a specific database, it may also accept an "INITIAL CATALOG=" item, to specify the name of that database, e.g. "INITIAL CATALOG=Customers".

In general it will be safer to use the previous syntax rather than this alternate one, but it is available if you want to try it.

## Sample ADO connect strings – using ODBC drivers without a DSN

---

### Microsoft Access databases

```
"PROVIDER=MSDASQL;DRIVER={Microsoft Access Driver (* .mdb) };DBQ=C:\MyAccessDatabase.mdb;UID=jsmith;PWD=secret;"
```

There is also an alternate way to specify the full path to the database file, as a separate folder and filename:

```
"PROVIDER=MSDASQL;DRIVER={Microsoft Access Driver (* .mdb) };DBQ=MyAccessDatabase.mdb;DefaultDir=c:\;UID=jsmith;PWD=secret;"
```

### Microsoft Excel workbooks

Note: If you have a license for the Formate Excel module, you can use functions such as `XIOpenWorkbook()` and `XIGetValues()` to read data from an Excel workbook, which is generally more reliable, more flexible, and simpler to set up than using external access via an ODBC driver.

```
"Driver={Microsoft Excel Driver (*.xls) };DBQ=c:\MyExcelWorkbook.xls"
```

Note: The remaining comments about the Excel connect string apply equally, regardless of whether or not the string uses a DSN.

An ODBC connection to Excel is read-only by default. You must add `"ReadOnly=0"` to your connect string if you want to make changes to the data. Otherwise, you will receive the following error message: "Operation must use an updateable query".

By default, it's assumed that the first row of your Excel data source contains column headings which can be used as field names. If this is not the case, you must turn this setting off, or your first row of data "disappears" to be used as field names. Do this by adding `"FirstRowHasNames=0"` to the connect string. The driver then names your fields F1, F2, and so on.

Note: due to a bug in some early versions of the ODBC driver, the `"FirstRowHasNames"` setting has no effect. In other words, the Excel ODBC driver (MDAC 2.1 and later) always treats the first row in the specified data source as field names. For additional information on this bug, see Microsoft Knowledge Base article 288343.

Excel does not provide ADO with detailed schema information about the data it contains, as a relational database would. Therefore, the driver must scan through at least a few rows of the existing data in order to make an educated guess at the data type of each column (i.e. is it numeric, currency, date, plain text, etc). The default is for it to scan 8 rows. You can change this to anything between 1 and 16 rows, by adding `"MaxScanRows="` to the connect string. For example, `"MaxScanRows=16"`. Or you can use `"MaxScanRows=0"` to force Excel to scan all existing rows.

Note: due to a bug in some early versions of the ODBC driver, specifying `"MaxScanRows="` has no effect. In other words, the Excel ODBC driver (MDAC 2.1 and later) always scans the first 8 rows in the specified data source in order to determine each column's data type. For additional information about this, see article 189897 in the Microsoft Knowledge Base.

If the Excel workbook is protected by a password, you cannot open it for data access, even by supplying the correct password with your connection settings, unless the workbook file is already open in the Microsoft Excel program. If you try, you will receive the following error message: "Could not decrypt file".

See article 257819 in the Microsoft Knowledge Base for general information about using the ODBC driver to access Excel workbooks.

### Microsoft SQL Server databases

```
"PROVIDER=MSDASQL;DRIVER={SQL Server} ;SERVER=MySQLServer;DATABASE=Customers; UID=jsmith;PWD=secret;"
```

### Microsoft Visual FoxPro (with a database container)

```
"Driver={MicrosoftVisualFoxProDriver};SourceType=DBC;SourceDb=c:\MyFoxproCo  
ntainer.dbc"
```

### Microsoft Visual FoxPro (without a database container)

```
"Driver={MicrosoftVisualFoxProDriver};SourceType=DBF;SourceDb=c:\MyFoxproDa  
tabase.dbf"
```

### Oracle databases

```
"PROVIDER=MSDASQL;DRIVER={Microsoft ODBC for  
Oracle};SERVER=MyOracleServer;UID=jsmith;PWD=secret;"
```

The "SERVER=" argument requires a value that matches the name of a service specified in the SQL Easy Net utility .

### Paradox databases

```
"Driver={Microsoft Paradox Driver (*.db)};DBQ=c:\MyParadoxDatabase.db"
```

### Text files

```
"Driver={Microsoft Text Driver (*.txt;*.csv)};DBQ=c:\MyTextFile.txt"
```

## ADO queries

---

### General

Once you have established a connection to your data source, you then need to query it so that you can obtain data. To do that, you must create a query string which tells the source what data you want, what criteria are to be used to select the appropriate records, and what order to return it in (if more than one record is to be returned).

The contents of that query string will vary somewhat according to what type of data source you are accessing. Most sources can accept a string that uses some version of SQL (Structured Query Language), a language which is purpose-designed for accessing databases. It's therefore always worth trying the SQL approach if you aren't sure exactly what is required (there are many good books available on SQL if you aren't already familiar with it). Some sources have a completely different language of their own, in which case, consult the documentation for the source (which is always a good idea in any case).

In addition to the query language, you must understand the data structure of the source you are querying, and very little help can be given here on that subject, because the structure will often be specific to your own environment (e.g. if you have a custom-created database for customer management). For example, to query a database, in addition to the name of the database itself you would need to know the names of the tables it contains, and the names of the fields within those tables. You would also need to know how the tables are related to one another (e.g. the "CustomerID" field in the "SalesOrders" table links each sales order to a specific record in the "Customers" table).

Without at least a basic understanding of the data structure, even if you manage to retrieve some data it may not be the data you wanted (e.g. you ask for total sales value for a particular customer but accidentally get the total sales value for all customers combined instead). If you aren't familiar with the data structure, you either need documentation on it or access to support staff who are familiar with it.

This remainder of this section covers a few basic points about some of the more commonly used sources, to get you started. It is not meant to be an exhaustive treatment of the subject because there are too many data sources available for that to be practical, so specific documentation for the data source you are using should always be consulted if possible.

### Microsoft Active Directory

#### General

Active Directory is software on recent Windows-based servers which stores information about the users, machines, printers and other resources available in an organisation. Having all the information together in one place is aimed at making it easy to manage, and simple for users to locate a resource when they need it.

There may be times when you need to look up information from Active Directory (AD), for example to find out what printers are available. You can essentially query any part of the AD structure, finding out which objects are present, and what the properties of those objects are.

AD can be queried using standard SQL syntax:

```
"SELECT objectCategory,name,ADsPath FROM  
'LDAP://OU=Volumes,DC=acme,DC=co,DC=uk' WHERE objectClass='volume'"
```

The following sections cover the syntax in a little detail and describe some parts which are specific to Active Directory, but any good book on the SQL query language will give plenty of background about how to build up a query in general.

#### SELECT

Between the words "SELECT" and "FROM" is a list of the properties you want to return from whatever objects the query finds. Multiple properties can be specified (as in the example), separated by commas.

For some reason these are normally returned in the reverse of the order in which you specified them, so keep that in mind (e.g. if you ask for "name,rank,number", in the results you will get number in column 1, rank in column 2 and name in column 3).

### **FROM**

After the word “FROM”, the next part must identify which area of AD you want to search, and it must be surrounded with a single quote mark (apostrophe) at either end. It always starts with “LDAP://” to identify the source as one which will be accessed through Lightweight Directory Access Protocol, which is the protocol ADO uses to query Active Directory. This is followed by a complete identifier for the “container” within which you want to search, or for the object whose properties you want to return. AD has a tree-like structure and so it works in a similar way to folders and files on your hard disk. If the identifier specifies a container (which is similar to a folder), the query will look at the objects which are within that container. If the identifier specifies an object which is not a container (which is therefore similar to a file), the query will return the properties of just that one object.

The identifier is given as a series of elements, rather like the elements you use when identifying a file on your hard disk, e.g. “c:\program files\formate\formate.exe”. In that case “c:”, “program files” and “formate” are all effectively folders (equivalent to containers in AD), while “formate.exe” is a file (equivalent to a non-container object). The elements of the path are separated by backslashes.

Identifiers are given in a similar way by building up a list of elements which shows how you get from the top of the Active Directory tree to the object or container you want, but there are a few differences in the format. Firstly, the elements are given in reverse order, starting with the object or container you want then specifying how to get from there back to the top of the tree, one stage at a time (file paths start at the top of the tree and work down to the file you want). Also, the separator between the elements is a comma rather than a backslash. Finally, each element must include a prefix and equals sign, where the prefix identifies the type of the element.

There are various element types you can use in the identifier:

**DC** – This comes from “Domain Controller”, but is actually used for any part of a domain name. For example, in “trilogic.co.uk”, each part of the domain name is a separate DC element, so it would be given as “DC=trilogic,DC=co,DC=uk”. For “trilogic.com” it would be “DC=trilogic,DC=com”. If your domain is further split down, for example into departments, you may need to add another DC element before the rest, e.g. “DC=Accounting,DC=trilogic,DC=co,DC=uk”.

**OU** – An “Organizational Unit”, such as a group of users within the company (e.g. “Main Office”) which has been added to Active Directory as a container for all objects related to that group. Something like a department could have been set up as part of the domain name (as shown in the previous paragraph) so that each department is in its own domain, and in that case the DC type is the one to use for it. If, on the other hand, your company has only one domain containing all departments, then the departments may well have been set up as Organizational Units within the domain, in which case use OU for the departments. Your system administrator should know which way it was done, or you could just try OU and DC in turn and see which works.

**CN** – The “Common Name” of an object. Use this when referring to an object where none of the other element types applies.

A typical identifier for the “Volumes” container at “acme.co.uk” might be:

```
“OU=Volumes,DC=acme,DC=co,DC=uk”
```

If you want to query every object in your domain rather than one specific container or object, just supply the DC parts, for example:

```
“DC=acme,DC=co,DC=uk”
```

### **WHERE**

After the identifier, you can optionally include the word “WHERE” followed by criteria to restrict the results to objects that meet certain conditions. For example, each object has a property called “objectClass”, which tells you what type of object it is. You could thus use the following to restrict the query to returning only “user” objects:

```
“WHERE objectClass='user' ”
```

Note that any literal strings you use must be enclosed in a pair of single quote marks (apostrophes).

You can combine multiple criteria using “AND” and “OR”, for example:

```
“WHERE objectClass='user' OR objectClass='computer'”
```

```
“WHERE objectClass='user' AND name='John Smith'”
```

If you want to combine “AND” and “OR” in a single query, take care with the order in which you use them because there is the possibility that the you may otherwise retrieve the wrong data. For example, consider the following, where you are trying to retrieve two objects, both called “Jim”, one being a user record and the other being the record for his computer (using the same name for both is not an advisable practice, but serves to illustrate the point):

```
“WHERE name='Jim' AND objectClass='user' OR objectClass='computer'”
```

You might be surprised to find that every computer on the system is returned, rather than just the one called “Jim”. This happens because the “AND” is evaluated first, so ADO first looks for objects where “name='Jim' AND objectClass='user'”, then adds any others where “objectClass='computer'”, i.e. all the computers. To avoid this, you can use brackets around the parts that need to be evaluated together, e.g.

```
“WHERE name='Jim' AND (objectClass='user' OR objectClass='computer')”
```

In addition to the equals sign which looks for exact matches, you can also use “<” to find objects where a property is less than a given value, or “>” to find ones where the property is greater than a certain value. Similarly, “<=” and “>=” look for ones less than or equal to, and greater than or equal to respectively. Use “<>” to search for objects where the property is anything other than the given value (i.e. it’s “not equal to”). Note: In this context, for text properties (i.e. non-numeric ones), “less than” means earlier alphabetically, and “greater than” means later alphabetically.

If you want to look for objects that have a property similar to a given value rather than exactly the same, you can use wildcards. These are special characters which represent a part of the text where you don’t care what the property contains, you still want it to match. The asterisk is the most commonly used wildcard, which represents zero or more unspecified characters at the point where it appears. For example, “P\*” would match all objects whose property starts with a “P” followed by zero or more other characters. “\*le” would find all those which end in “le”, while “\*the\*” would find all those which have “the” anywhere within the text, including at the start or end, or even if the text consisted solely of “the”. Use wildcards only with the equals sign (the usual SQL keyword “LIKE” cannot be used here), e.g.

```
“WHERE name=' P* ’”
```

### ***Objects and properties***

Active directory supports various types of object, including “Computer”, “Contact”, “Group”, “Organizational Unit”, “Printer”, “User” and “Shared Folder”.

Each has its own set of properties, such as a name, an object class, perhaps an email address and so on. Some properties are found in all object types, such as the name, but some properties only apply to the relevant type of object.

It’s possible to extend the schema of your Active Directory system to include new classes of object with customised properties, so it isn’t feasible to cover every available object type and property here because every site could be different. If you need to access customised objects, contact their designer for information about what classes and properties they have.

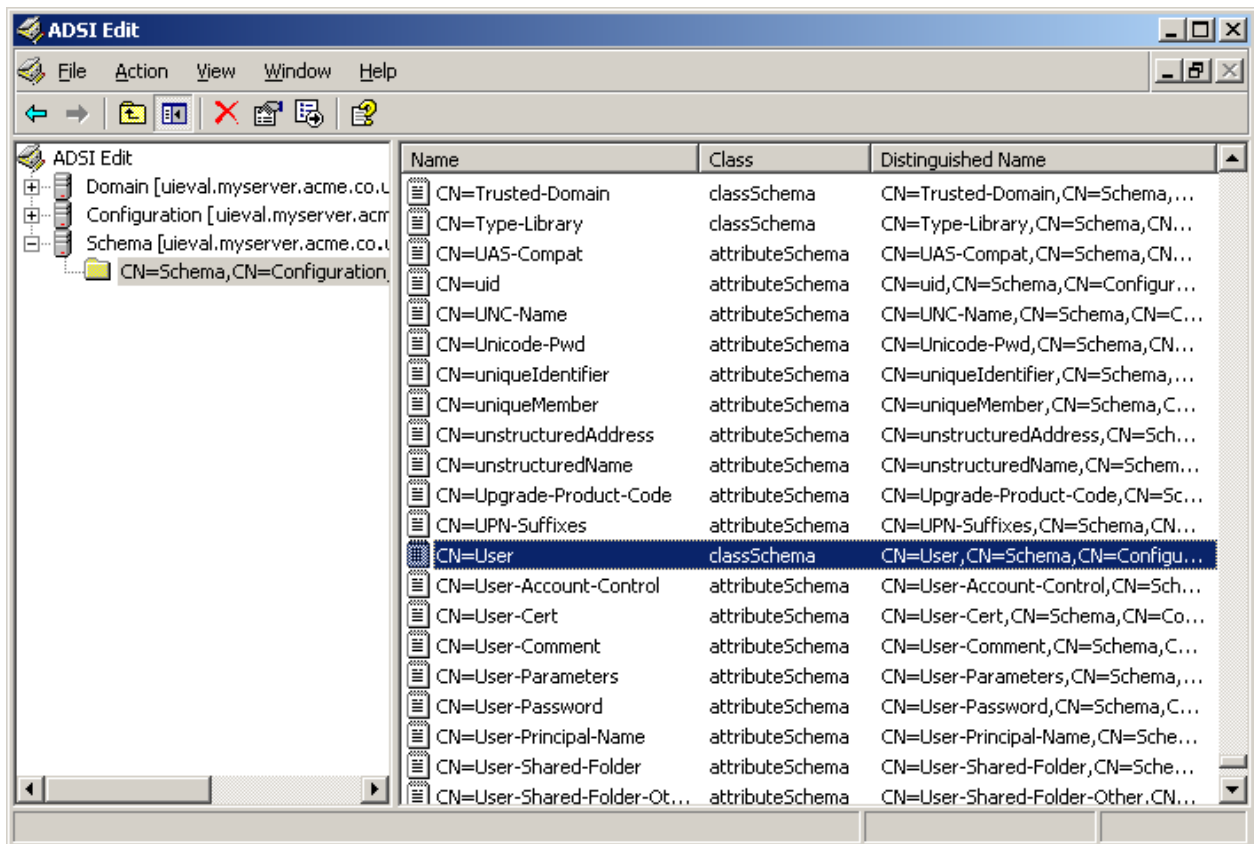
You can use the names of properties either in the SELECT list to specify which information you want to retrieve (e.g. to get the name of the object), or as part of the WHERE criteria (e.g. to find objects with a particular name or email address).

You can also use the same property in both the SELECT list and the WHERE criteria in the same query if you want to. Generally it isn’t necessary because if you are restricting the query to only return objects with a specific name, for example, you already know what the name of every returned object will be so there is no need to ask Active Directory to return it, and the query will run slightly faster if you don’t. There may be times, however, when you’re using criteria that don’t mention an exact name (e.g. where they include wildcards to return every name starting with an “S”), or when you are using two sets of criteria connected by an OR so you don’t know which set matched the object. In that case, you may need to include the property in the SELECT list as well as the WHERE, so that you’ll know what value it has on each returned object.

Some properties have multiple values for each object. For example, “objectClass” – each object can be a member of several classes, so when you use it in a selection criterion, it matches if any of the classes is the one you requested. Unfortunately, this causes problems for ADO because it can’t return multiple values in a single field, so if you try to return “objectClass” you will either get an error or a blank value. If you come across a property which fails in this way, it’s most likely for that same reason, and therefore the property cannot be returned and isn’t accessible. Often there is a related or equivalent one you can use instead (“objectCategory” could probably be used instead of “objectClass” in many cases). This issue doesn’t generally prevent you using it in your criteria, it just means you can’t return the value in the query results.

### *How to find out what properties are available*

As mentioned in the previous section, it is outside the scope of this document to try to cover every available property, so you may need to find out for yourself which ones are present in each class of object. To do that, you need the ADSIEDIT snap-in for MMC (Microsoft Management Console). This snap-in allows you to inspect each of the classes in the schema, to see what “attributes” (i.e. properties) it has.

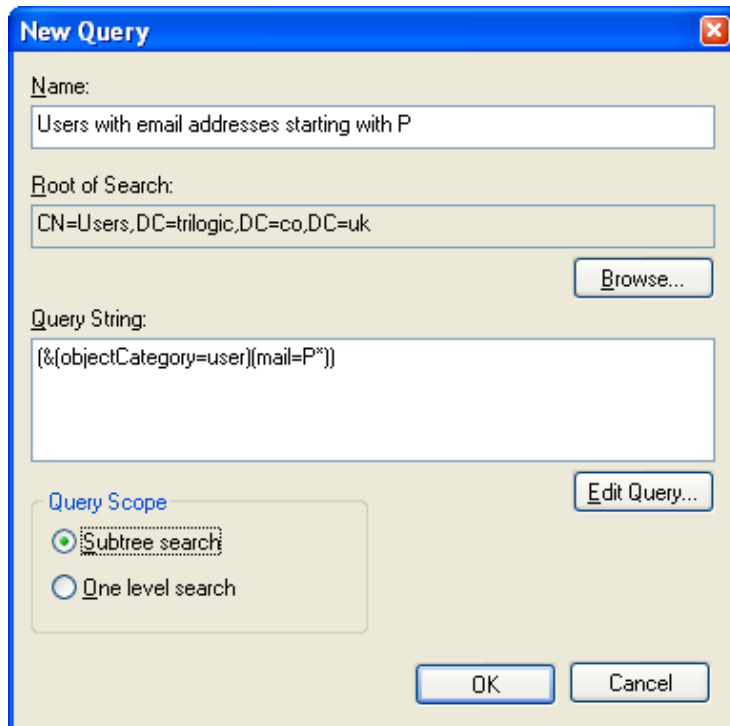


If you don’t already have the ADSIEDIT snap-in installed, it is available in the “\support\tools” folder on the Windows Server 2000 or Windows Server 2003 CD. **Either** install all the support tools using the provided Setup program, **or**:

1. Navigate to the “\support\tools” folder.
2. Double-click on the Support.cab file.
3. Locate the files adsiedit.msc and adsiedit.dll. Extract them to your “windows\system32” folder.
4. Open a command window in the “windows\system32” folder and run the command “regsvr32 adsiedit.dll” to register the dll.

Once the snap-in is installed, start Microsoft Management Console (MMC), and add the ADSI Edit snap-in. Alternatively, you can set up a shortcut on the desktop which executes “adsiedit.mmc”.

Once the snap-in is running, right click the “Domain” item in the tree, then select “New”, “Query” to bring up this dialog:



Enter a suitable name for the query, then use the “Browse” button to select which part of the Active Directory structure you want to search, and set “Query Scope” appropriately (use “Subtree” if you want to search the contents of other containers inside the one you selected, otherwise use “One level” to search only the selected container itself).

Use “Edit Query” to fill in the query string. This pops up a dialog which lets you build a query string by selecting from lists of available properties, possible comparison operations, etc. This has the great advantage that the dialog knows which properties are available on each type of object, and exactly what their names are, so the resulting query string should always be a valid one

Once the query is complete and saved, the snap-in adds it to the tree structure and any objects it returns appear as sub-items under it. Handily, this allows you to test the query to make sure it returns the objects you expected and doesn’t produce errors.

It’s normally simpler to create queries for Active Directory using the SQL-style syntax described earlier because many people are already familiar with it, and if you wish to do that you can just use this dialog in order to find out what properties are available and exactly what their names are. To use them, you need to copy the name(s) from the Query String and paste them into the SELECT and/or WHERE parts of your SQL query string in Formate as applicable.

However, the dialog actually contains large parts of the query, ready to use in an alternate LDAP-specific syntax which is covered in more detail in the next section, so you may want to use that syntax instead. It’s more complex and you may not be familiar with it, but the dialog has already done the difficult parts, so it’s basically copy and paste from here on.

### *An alternative LDAP-specific syntax*

This alternate syntax is a standardised one used to query LDAP (Lightweight Directory Access Protocol) databases over the Internet, not just Active Directory. The full specification may be found in a number of Internet RFCs (Request For Comment) – see Microsoft Knowledge Base Article 221606 for a list of the relevant RFCs.

To use the LDAP-specific syntax, your query should take the form:

```
"<LDAP://Myserver/Root>;QueryString;PropertyList"
```

Where "Myserver" is the machine name of the Active Directory server, and "Root" is the contents of the "Root of Search" field from the dialog. "QueryString" is the value from the "Query String" field in the dialog, copied and pasted into the string completely unaltered. "PropertyList" is a list of the properties you want the query to return, each one separated by a comma. You do need to construct this last part manually, but you can refer back to the dialog to see what properties are available, because they are the same ones available in the pop-up lists when you are building the query string.

A complete string to query the "Users" container on the server "Server1" in the "acme.co.uk" domain might therefore be:

```
"<LDAP://Server1/CN=Users,DC=acme,DC=co,DC=uk>;(&(objectCategory=user)(mail=P*));name,mail,whenChanged "
```

This would search for user records with an email address starting with "P", and return the user's name, email address, and the date when the user record was last changed.

You probably wouldn't need the criterion restricting "objectCategory" to "user" if you were only looking in the "Users" container because the contents are all likely to be user objects in any case, but it illustrates the way in which more complex criteria may be built up.

The "QueryString" part of the query is optional, and may be omitted if you just want to return every object in the specified container. In that case you must still include the semi-colon separators that would normally be around it, even though there is nothing between them, e.g.:

```
"<LDAP://Server1/CN=Users,DC=acme,DC=co,DC=uk>;;name,mail,whenChanged "
```

Unlike when using the SQL syntax, the properties you select generally seem to be returned in the same order in which you requested them, not the reverse order. For example if you ask for "name,rank,number", in the results you will get name in column 1, rank in column 2 and number in column 3, as you would probably expect.

### Microsoft Excel workbooks

The Excel provider allows you to use a SQL-like syntax, so the language itself is fairly simple. What you do need to be aware of, however, is that because Excel doesn't store data in fields and tables as a database would, you have to specify which data you want to retrieve in a way which is specific to Excel.

You can retrieve Excel data from any of the following: an entire worksheet, a named range of cells, or an unnamed range of cells. Note that if you retrieve more than one row of data, you must use DbFillArray() rather than DbLookup(), as the latter can only handle one row.

#### *Retrieving entire worksheets*

To retrieve an entire worksheet, use the worksheet name followed by a dollar sign and surrounded by square brackets. For example, to return the entire contents of "Sheet1" use:

```
"SELECT * FROM [Sheet1$]"
```

If you omit both the dollar sign and the brackets, or just the dollar sign, you receive the following error message: "The Jet database engine could not find the specified object". If you use the dollar sign but omit the brackets, you will see the following error message: "Syntax error in FROM clause."

The provider assumes that your table of data begins with the upper-most, left-most, non-blank cell on the specified worksheet. In other words, your table of data can begin in Row 3, Column C without a problem, and that will be the first cell returned. However, you cannot then, for example, type a worksheet title above and to the left of the data in cell A1.

#### *Retrieving a named range of cells*

To specify a named range of cells, simply use the name you gave to the range. For example:

```
"SELECT * FROM MyRange"
```

### *Retrieving an unnamed range of cells*

To specify a range of cells which hasn't been named, append standard Excel row/column notation to the end of the sheet name in the square brackets. For example:

```
"SELECT * FROM [Sheet1$A1:B10]"
```

### **Adding new rows**

When you specify a worksheet as your source and you add new rows, the provider adds them below the existing records in the worksheet as space allows.

When you specify a range (named or unnamed), it also adds new records below the existing records in the range as space allows. However, if you query on the original range, the resulting data does not include the newly added records which are outside that original range.

With the provider supplied with MDAC (Microsoft Data Access Components) versions prior to 2.5, when you specify a named range you cannot add new records beyond the defined limits of the range, or you receive the following error message: "Cannot expand named range".

### **SQL databases**

In general, these use nothing more than standard SQL (Structured Query Language) to return data and even to carry out updates to the database, so any good treatment of SQL may be consulted to find out how to create a query string for them. Even some databases which do not strictly use a SQL Database Management System (e.g. Microsoft Access) can still be queried using a syntax which largely follows the SQL standard.

You will also need to know the specifics of the data structure you are querying, e.g. what table names and field names are in use, how tables relate to one another, and so on. This can be found by reading any documentation you may have on it, or by inspecting the database if you have a program which lets you see the database structure (e.g. Microsoft's SQL Enterprise Manager, or even MS-Access can be used). Otherwise, you will need to contact the provider of the database for assistance.