

## Quickstart guide to ADO

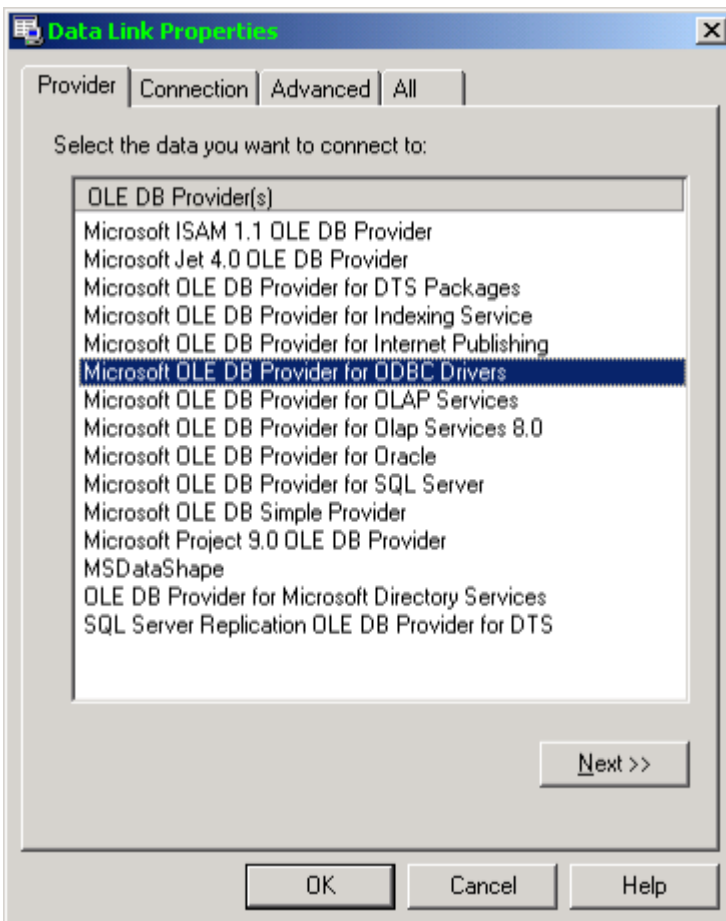
ADO (ActiveX Data Objects) is a relatively new method to read and write data from and to databases. The main advantage of using ADO is the fact that you don't need an ODBC alias to connect to a database.


ADO uses a connection string to connect to a database. This allows the Enterprise Dynamics user to define the connection from within Enterprise Dynamics.

### Step 1: Creating the connection string

The connection string, depending on the type of database, can be very long and complex. However a 4DScript function is available to return the connection string to a specific database: [ADOBuildConnectionString](#)

This will result in a wizard that helps you to define the connection string.



 If you want to connect to a Microsoft Access® database, use the Microsoft Jet 4.0 OLE DB Provider.

After you have selected your database a connection string is returned that you can use to connect the database.

#### Title Text

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Enterprise Dynamics  
8.0\Work\Examples\Databases\My First Database.mdb;Persist Security Info=False
```

As you can see the connection string is quite long, but by using the [Concat](#) function you should be able to dynamically build a connection string based on a specified file name.


### Step 2: Connect to the database


Connecting to the database is easy.

## Connect to the database

```
ADODConnect ([Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Enterprise Dynamics 8.0\Work\Examples\Databases\My First Database.mdb;Persist Security Info=False])
```

This will open the database. Notice that the only parameter of `ADODConnect` is the connection string.

 This function also has an optional parameter to directly open a table for reading and writing. Write the table name between [ ].

 In the remainder of this quickstart guide we assume that you use the database “My First Database.mdb” that can be found in the examples\databases directory of Enterprise Dynamics.

## Step 3: Reading data from a table

The function `ADOSetTable` allows you to pinpoint a table from which you want to read data (you can also use this function for writing data):

### Activate a table

```
ADOSetTable ([ColorTable])
```

Now you have opened this table and the table is ready/has been made active.


You can read the data by entering the row number and the field number in the `4DScript` function `ADODReadTable`:


### Read the first field value from the first row

```
ADODReadTable (1, 1)
```

This will read the first field value from the first row. An optional third parameter of this function allows you to define how the value is returned:

- 0: anything
- 1: as a value
- 2: as a string

 If you want to copy all the rows from the database table to an Enterprise Dynamics table you can use `ADODTableCountRecords` and `ADODTableFieldCount` to determine the size of the table you want to copy it to.

 Be aware of the fact that the field definition in your database must match the value you want to return. Otherwise unexpected values or strings can be returned due to the fact that it can't be converted to a value or string.

## Step 4: Writing data to a table

In many cases a database is used to write output results. You can either edit existing data or you can add/append data to a table.

To edit an existing value use the following statement:

### Edit field number 2 from row 1 into the value 999

```
Do (  
  ADODTableEdit (1,   
  ADODWriteTableValue (2, 999),   
  ADODTablePost  
)
```

Notice that you need to position the database cursor to the row via the `ADODTableEdit` function. Next to that you

can change several values (or strings) in that specific row. You commit your changes by using the [ADOTablePost](#) function.


Adding a new row is done in the following way:

#### Adding a new row

```
Do (  
    ADOTableAppend,  
    ADOWriteTableValue(1, 345),  
    ADOWriteTableValue(2, 999),  
    ADOTablePost  
)
```

In this example you can see that multiple values are written to the newly created row and after the values have been written the row is committed to the database by using [ADOTablePost](#). This behavior is not the same as with using databases via ODBC in Enterprise Dynamics. The reason for this is twofold:

- The ODBC connection uses an intermediate component that handles these kinds of things. But it actually does the same as the above example.
- The ADO connection is much faster than the ODBC connection by keeping it lean. By doing so you need to “post” the new data to the table yourself.

 To write a string to a table use the 4DScript function [ADOWriteTableText](#).

#### Step 5: Using queries

Queries allow you to make a selection of data from one or more tables. It is therefore not possible to edit or append data. However it is possible to pass an update query.


To select a result set based on a query use the 4DScript function [ADOSetQuery](#).

#### Select a result set based on a query

```
ADOSetQuery([Select * from ColorTable])
```

This will result in all the records from the table `ColorTable`.

In the same way as you read data from a table you can now read data from the resultset of this query by using [ADOReadQuery](#). All the parameters are the same as [ADOReadTable](#).

 To find out the number of rows and the number of fields of the resultset of your query use [ADOQueryCountRecords](#) and [ADOQueryFieldCount](#).

If you would use the function [ADOSetQuery](#) in the following way you would pass an update query that will result in adding a new record, in this case, the `ColorTable` table.


#### Pass an update query that will result in adding a new record

```
ADOSetQuery([Insert Into ColorTable (ID, Color) Values (123, 345)])
```

Update queries can be very powerful because they perform data manipulation at the server side, which avoids the transfer of a lot of data through the network (or internet). However these queries can be a little bit more difficult to create. In most cases the database application itself will assist you in creating such a query through wizards.

#### Step 6: Closing the database

You can close the database by using the 4DScript function [ADOClose](#).

 If you want to open a second database you don't need to use the command [ADOClosefirst](#). [ADOConnect](#) will close the current database automatically before opening the newly selected database.



Opening a database is relatively slow in comparison with the execution of other 4DScript functions. It is recommended that avoid switching between databases.