

NeoDBpro

Table of contents

Welcome to NeoDBpro	3
Introduction to NeoDBpro	3
Creating Database Enabled Publications	4
Opening a Database	4
Creating a New Database	6
Working with Data	9
Special Variables	10
Picture Field Considerations	11
Defining Colors	13
Distributing Database Publications	13
NeoDBpro Tutorial	15
Action Command Reference	20
Database	21
Tables	24
Grids	33
Navigation	43
Search/Query	45
Advanced SQL	49
Sort/Index	50
Add/Modify	52
Advanced Add/Modify	54
Simple Reporting	55
Advanced Reporting	59
Import/Export	61
Stored Procedures	68
Math	71
Miscellaneous	73
Creating Reports	75
License Agreement	77

Welcome to NeoDBpro



NeoDBpro is an advanced, professional database plug-in designed for use with **VisualNEO for Windows**. NeoDBpro provides VisualNEO for Windows authors with the ability to add advanced database functionality to their publications. NeoDBpro uses ODBC and ADO technology to access a variety of database formats including Microsoft® Access™, MySQL™, SQLite™, Microsoft® SQL Server™, Oracle™, PostgreSQL™, Paradox, dBase and more.

[Introduction to NeoDBpro](#)

[NeoDBpro Tutorial](#)

[Creating Database Enabled Publications](#)

[Action Command Reference](#)

[Distributing Database Publications](#)

[Technical Support](#)

[Creating Reports](#)

[License Agreement](#)

Created with the Standard Edition of HelpNDoc: [Easily create EPub books](#)

Introduction to NeoDBpro

NeoDBpro uses ODBC and ADO technology to access a variety of database formats including Microsoft® Access™, MySQL™, SQLite™, Microsoft® SQL Server™, Oracle™, PostgreSQL™, Paradox, dBase and more.. In fact, when properly configured with the appropriate ODBC driver, NeoDBpro can be used to access just about any type of database system. Of course, database programming is a not trivial endeavor so don't expect to master it overnight.

What is a Database?

Before starting with NeoDBpro, it's probably a good idea to explain exactly what makes up a modern database. A database is a special type of file designed for storing information. An address book or a box containing recipes printed on index cards are examples of databases you've probably used many times. On a computer, database files are organized into tables which are organized into records. In turn each record contains a group of fields. If the address book were a computerized database, each person would be a unique record. Those records would each contain fields for the person's name, street, city, state, zip code and telephone number. Modern databases can be simple, like an address book, or they can be relational. A relational database is one that consists of multiple tables containing a collection of related data.

What are ODBC and ADO?

ODBC (Open Database Connectivity) is a standard database access method developed by the SQL Access Group. The goal of ODBC is to make it possible to access any data from any application, regardless of which database management system (DBMS) is handling the data. ODBC manages this by inserting a middle layer, called a database driver, between an

application and the DBMS. The purpose of this layer is to translate the application's data queries into commands that the DBMS understands. For this to work, both the application and the DBMS must be ODBC-compliant - that is, the application must be capable of issuing ODBC commands and the DBMS must be capable of responding to them.

ADO (ActiveX Data Objects) is Microsoft's high-level interface for data objects. ADO is designed to access all sorts of different types of data, including databases, spreadsheets, and other types of documents. Together with ODBC, ADO is one of the main components of Microsoft's Universal Data Access (UDA) specification, which is designed to provide a consistent way of accessing data regardless of how the data are structured.

Created with the Standard Edition of HelpNDoc: [Write eBooks for the Kindle](#)

Creating Database Enabled Publications

Getting Started

Your first step depends on what type of database you're planning to use and where it's stored. Some types of databases are stored in traditional files, like Microsoft Access, and can simply be opened. Other database types, like MySQL or Oracle, require that you connect to a database server. You can think of the server as the engine that handles everything to do with the database. Often these types of databases will require special software and database drivers to be installed on your computer.

[Opening a Database](#)

[Creating a New Database](#)

[Working with Data](#)

[Special Variables](#)

[Picture Field Considerations](#)

DISCLAIMER: If you are operating in a corporate or organization environment, you must obtain permission from the database administrator before attempting to access any database with NeoDBpro or any other tool. Please be extremely careful if you are experimenting with a live, mission critical database. If the database doesn't have the proper security precautions, it's certainly possible for someone who doesn't know what they are doing to delete important data or otherwise cause serious damage to a database.

Created with the Standard Edition of HelpNDoc: [Easy CHM and documentation editor](#)

Opening a Database

In order to connect to a database server you will need to know some specific information such as the name of the database, your user name and a password. This information, called a **Connection String**, must be formatted precisely or you will not be allowed to access the data.

Installing the proper drivers and configuring the server **Connection String** is often the most challenging part of working with ODBC/ADO databases, because each type of server requires slightly different information in order to establish a connection. Fortunately, NeoDBpro includes some wizards to make the process of connecting to most types of popular databases easy than it usually would be.

Opening a Microsoft Access (MDB or ACCDB) Database

The easiest to configure, and the most portable type of database, is Microsoft Access. As

described above, most other types of databases require special software and drivers to be installed on your computer. The Microsoft Data Access Drivers (MDAC) required for Microsoft Access databases are included with Windows starting with 98. Therefore publications that deploy Access databases should be viewable on all but the very oldest Windows 95 computers. (Windows 95 users can download MDAC free from Microsoft's web site if needed.)

To open an existing Access database, simply pass the file name to NeoDBpro's [dbpOpenAccessDatabase](#) action. For example, opening a file called "AddressBook.mdb" located in the "c:\my documents" folder would look like this:

```
dbpOpenAccessDatabase "MyDB" "c:\my files\AddressBook.mdb" ""
```

The first parameter "MyDB" is the **Database ID**, which is simply the name that you want to use to refer to this database in the future. It can be anything you like, but it's often easiest to use something short and descriptive. A Database ID is also sometimes referred to by database professionals as an "Alias".

The second parameter is the database file name and the third parameter is the database password if required.

Opening Other Types of Databases

To open or connect to most other types of databases, you will use an action called [dbpOpenDatabase](#) which requires something called a **Connection String**.

The connection string is used to specify the information needed to connect to the database server. Connection strings can be tricky because they are often different depending on what type of server/ODBC driver you're using. The string consists of one or more elements required to establish the connection. Specify multiple elements as a list with individual elements separated by semicolons. For example, the command to open a MySQL database looks like this:

```
dbpOpenDatabase "MyDB" "Provider=MSDASQL.1;Driver={MySQL ODBC 3.51  
Driver};Server=localhost;Database=test;User=root;Password=apple;Option=3"
```

The first parameter "MyDB" is the **Database ID**, which is simply the name that you want to use to refer to this database in the future. It can be anything you like, but it's often easiest to use something short and descriptive. The Database ID is also sometimes referred to by database professionals as an Alias.

The second parameter is the connection string. This particular connection string contains the unique elements required by MySQL. A different type of database might contain some of the same elements or completely different ones. Check your database's documentation to determine the appropriate connection string for your server. (See disclaimer [here](#)!) Your database administrator should also be able to provide you with an appropriate connection string. You can also find example connection strings by searching for "ADO connection string" on Google. Otherwise, you can just experiment with different connection strings to see what works.

Working with Very Large Databases

By default NeoDBpro uses something called a **client-side cursor**. Simply put, this is a method of accessing data that relies on the client application to handle most of the data storage and processing. Client-side processing is very flexible and generally provides the fastest performance for small to medium sized databases. For large databases, however, a client-side cursor can sometimes consume too many system resources resulting in poor performance. To compensate for this problem, you may want to use a **server-side cursor** when working with very large databases. In NeoDBpro you can switch to a server-side cursor by adding "CursorLocation=Server" to your connection string. For example:

```
dbpOpenDatabase "MyDB" "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=[PubDir]
AddressBook.mdb;CursorLocation=Server"
```

With a client-side cursor (the default) all data is copied to the local machine and processed there. This provides access to features not normally supported by servers such as sorting and indexing. When using a SQL query, only the data returned is copied to the local machine. A server-side cursor doesn't provide as much flexibility, but is often more appropriate for large databases, and may be required when the size of a database exceeds the available memory and disk space available on a local machine.

After Opening a Database

To tell if the dbpOpenAccessDatabase or dbpOpenDatabase action has worked, you can monitor the variable screen of VisualNEO for Windows's debugger. When NeoDBpro successfully connects to the database server, your debugger should display the variable below:

```
MyDB.$Status = Connected
```

MyDB is the Database ID you selected.

Once a connection has been established, you can open tables with the [dbpOpenTable](#) action, or perform a query by passing an SQL "SELECT" statement to the [dbpExecSQL](#) action. For example, to view the contents of a table named "Customers", use one of the following:

```
dbpExecSQL "MyDB" "SELECT * FROM Customers"
```

or

```
dbpOpenTable "MyDB" "Customers" ""
```

This will access all of the fields and records in the Security table. The first parameter is the Database ID name used in the dbpOpenDatabase action. The second parameter is the SQL statement. See [Working with Data](#) to see how NeoDBpro handles the actual data from the database.

Closing a Database

When you're finished reading from the database, you can close the connection using the [dbpCloseDatabase](#) action. For example:

```
dbpCloseDatabase "MyDB"
```

If you forget to close the connection, NeoDBpro will do it for you automatically when your publication shuts down.

Created with the Standard Edition of HelpNDoc: [Free EBook and documentation generator](#)

Creating a New Database

Creating Databases

The process for creating a new database from scratch varies depending on what type of database you're using. Some databases can only be created by using special software running on the server. Other databases can be created by simply creating a new folder on the server or local hard drive. For these reasons NeoDBpro doesn't offer options for creating all of the different types of databases. The exception is Microsoft Access format databases, which can be created easily using the special [dbpCreateAccessDatabase](#) action. For other types of databases consult your database product's documentation.

Creating Tables

Online databases, tables are realative easy to create and NeoDBpro provides the tools needed to create just about any kind of table you can imagine. Of course, building a new table from scratch is a little more complicated than opening one that's already been created, but NeoDBpro's [dbpCreateTable](#) action makes the process mostly painless.

To create a new table from scratch, you will need to decide on a name of your new table and provide some basic information about the fields you want it to contain. If a table with the name you specify already exists in the database, NeoDBpro will generate an error rather than overwrite it. (You can use the [dbpDropTable](#) action to remove exists table if you want to ensure that dbpCreateTable won't fail. Be sure to exercise caution when using dbpDropTable because any data the deleted table contains will also be deleted.

Next, you will need to create a list of the types of fields that will comprise each record in the new table. Each field definition consists of a unique field name, the field type and, depending on the field and database type, size and options. Multiple fields are separated by semicolons (;). For example:

```
"field1 type(size) options;field2 type(size) options;..."
```

Field names can be just about anything you like as long as each name is used only once in the same table. Typically a field's name should reflect the type of data it will contain, such as LastName, City, State, etc.

Tables store different types of data in specific types of fields. Text is stored in either a string, char or memo type field, while numbers are stored in an integer or float type field. Since VisualNEO for Windows doesn't really distinguish between text and numbers the way traditional programming environments do, you can get by with defining all of you fields as strings if you like. The only exception would be if you plan on loading your database file into another program. In that case, you may want to define your fields using the field types that more closely match your data.

Note: Unfortunately, different databases offer different, sometimes incompatible, array of choices when it comes to field or data types. Sometimes even field types with the same names can have different capacities or properties in another database. To be sure you're getting what you expect, check the database's documentation.

The types of database fields supported by NeoDBpro are listed below:

String	Variable-length character data with a maximum of 8,000* characters. Used for storing text or combinations of text and numbers, such as addresses. Can also be used for numbers that do not require calculations, such as telephone numbers, part numbers, postal codes, etc. Some types of databases refer to this field type as a "VarChar". The desired size of the field is specified in parenthesis. For example: String(500)
Char	Fixed-length character data with a maximum length of 255* characters. The desired size is specified in parenthesis. For example: Char(35)
Memo	Variable-length character data with a maximum length of 64,000* characters. This field type is sometimes called a "Text" field.
Boolean	Can contain either "true" or "false". This field type is sometimes called a Bit.
Integer	A whole number between -32,768 and 32,767*.
BigInt	A very large whole number between -2,147,483,648 and 2,147,483,647*.
AutoInc	A unique sequential number automatically inserted when a record is added. The number will be incremented by one for each new record.
Currency	Monetary values. Accurate to 15 digits to the left of the decimal point and 4 digits to the right. This field type is sometimes called a "Money" field.
Float	Decimal number between 10 ⁻²⁸ and 10 ²⁸ - 1*.

Date	Date. This type is not supported by MS Access, use DateTime instead.
Time	Time. This type is not supported by MS Access, use DateTime instead.
DateTime	Date and time data.
Picture	Can be used to store just about any type of binary data, including pictures, document files, etc. This field type is sometimes called "binary", "varbinary" and "blob".

Not all types of databases support this type of field, and there are no standard formats for storing data. This means that data stored here may not be easily exchanged between different types of databases. See [Picture Field Considerations](#) for more information.

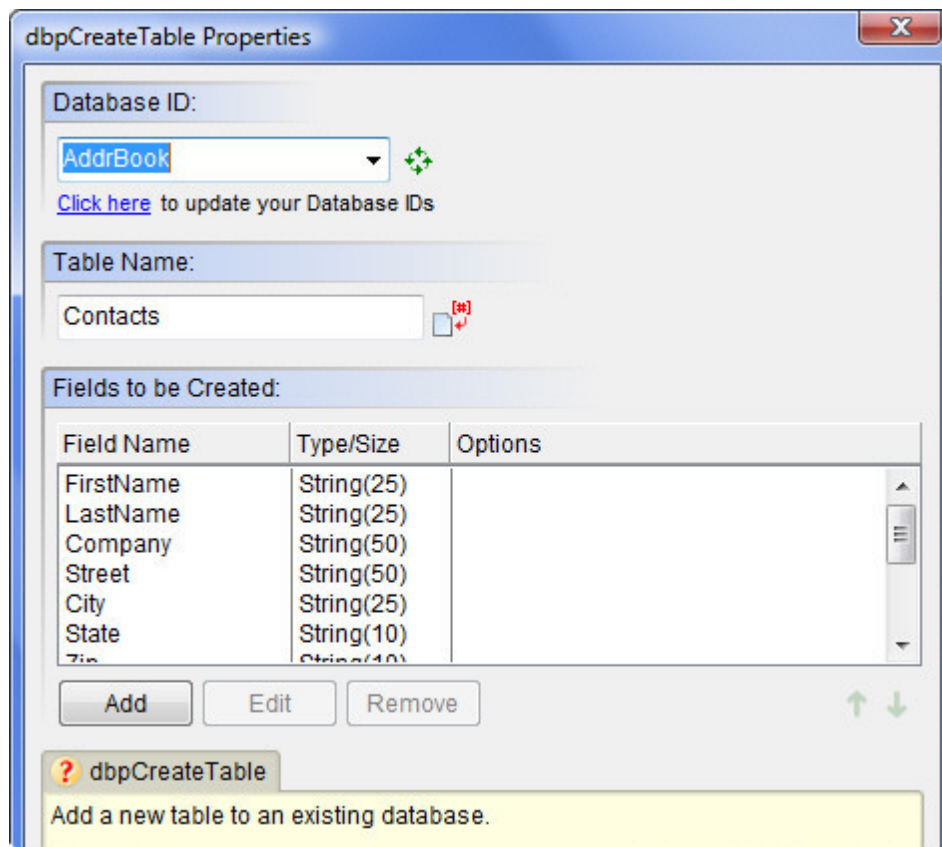
*Maximum field sizes vary widely between databases and even among versions of the same database. Check your database's documentation for exact field properties.

Finally, the field size is used to define the maximum length for String and Char types. Non MS Access databases can optionally specify sizes for Integer, BigInt and Float types. Size is ignored for all other field types. You can reduce the size of your database by specifying maximum sizes for each String and Char fields. For example, a zip code field rarely requires more than 5 characters and last names can usually fit in 25 characters. Allocating more space for these types of fields will make your database file larger than it needs to be.

Here's an example showing how to create a simple database consisting of first name, last name and telephone fields:

```
dbpCreateTable "AddrBook" "Contacts" "FirstName String(25);LastName
String(25);Telephone(30)"
```

Because dbpCreateTable can be complex, the easiest way to properly construct this syntax is to select the action from VisualNEO for Windows's Select and Action list and use the dbpCreateTable wizard pictured below:



After you define your table using the wizard you will be asked if you want NeoDBpro to create basic Text Entry boxes for each of your table's fields. If you answer "Yes" the Text Entry boxes will be placed onto the Windows clipboard. You can then return to VisualNEO for Windows and select Paste from Edit menu to place the objects onto your publication.

Note: Advanced users may choose to create tables by passing SQL commands to the [dbpExecSQL](#) action.

Created with the Standard Edition of HelpNDoc: [Easily create PDF Help documents](#)

Working with Data

Users of NeoSoft's entry level database plug-in, VisualNEO for WindowsDB, will find many of NeoDBpro's features and conventions familiar. But for those new to this realm, let's begin with a discussion about how NeoDBpro manages to get data from a database file into your VisualNEO for Windows publication.

As you know, a database can contain one or more tables, which contain records which are composed of fields. When creating or opening a database table, NeoDBpro will automatically create VisualNEO for Windows variables for each field it finds. Because NeoDBpro allows you to open up an unlimited number of databases and tables at the same time, and many tables use similar field names, NeoDBpro combines the database ID and the table's name with the field name to create a unique variable. For example, let's open a Microsoft Access database called "AddressBook":

```
dbpOpenAccessDatabase "MyDB" "c:\my files\AddressBook.mdb" ""
```

Next, let's open a table inside AddressBook called "Clients":

```
dbpOpenTable "MyDB" "Clients" ""
```

The Clients table contains the following fields:

```
FirstName
LastName
Street
City
State
Zip
Telephone
```

After opening the table, NeoDBpro would automatically create the following VisualNEO for Windows variables:

```
[MyDB.Clients.FirstName]
[MyDB.Clients.LastName]
[MyDB.Clients.Street]
[MyDB.Clients.City]
[MyDB.Clients.State]
[MyDB.Clients.Zip]
[MyDB.Clients.Telephone]
```

Each variable is a combination of the Database ID (MyDB), a period (.), the Table Name (Clients), a period (.) and the field name. Since NeoDBpro allows you to open multiple databases and tables at the same time, this method makes it unlikely that any two databases will have conflicting field names.

After opening the table, NeoDBpro copies the data from the first record into the corresponding field variables. If you've assigned these variables to Text Entry Fields or other

VisualNEO for Windows objects, you will see this data appear automatically on your screen when you run your publication. For example, each of the Text Entry Fields below is assigned one of the above variables:

The screenshot shows a window titled "DBPro AddressBook". Inside, there is a form with the following fields and values:

FirstName:	Frederick		
LastName:	Fishburn		
Street:	7986 Gabrielle Ln		
City:	State/Prov:	Zip:	
Aurora	IL	60598	
Telephone:	555-1212		

Only one record in the table is considered active at a time. This active, or current record, is the source of the field variables' data. If another record becomes active, the field variables are automatically updated with the contents of the new record. This linkage works both ways, so any changes to the field variables in VisualNEO for Windows are also automatically reflected in the database. This powerful link between the database and VisualNEO for Windows means that you don't need to be concerned with the complexities of reading and writing data.

Add navigation Push Buttons that call [dbpPrev](#) and [dbpNext](#) actions, and you've created a working database application using only a few NeoDBpro commands.

Created with the Standard Edition of HelpNDoc: [Easily create EBooks](#)

Special Variables

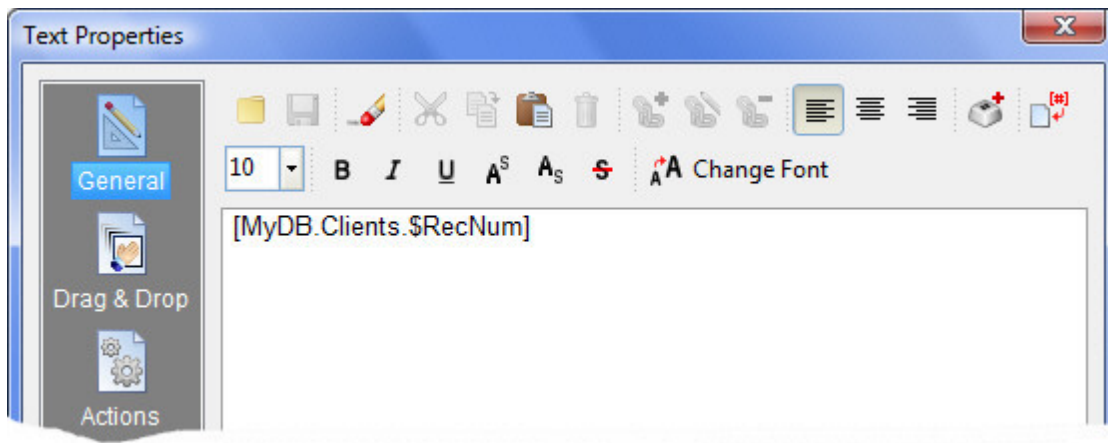
NeoDBpro allows you to open an unlimited number of database files and tables. Each database and table automatically create some unique global variables that contain important status information that you can access from VisualNEO for Windows.

As with field variables, each status variable is a combination of the Database ID, a period (.), the Table Name, a period (.) and the field name. Since NeoDBpro allows you to open multiple databases and tables at the same time, this method makes it unlikely that any two databases will have conflicting status variable names.

Status variables may be inserted wherever normal VisualNEO for Windows variables are accepted. For example, after calling `dbpOpenTable`, we can display an alert box to report the total number of records the table contains using the `$RecCount` variable:

```
dbpOpenTable "MyDB" "Clients" ""
AlertBox "Report" "The Clients table contains [MyDB.Clients.$RecCount]
records."
```

You could also display the number of the current record on screen by adding the `[MyDB.Clients.$RecNum]` variable to a VisualNEO for Windows Text object. For example:



Status variables are read-only, meaning that they can be displayed but not modified using the SetVar or any other action commands. Instead use NeoDBpro actions like dbpNext, dbpPrev, etc. to display a different record.

Below is a list of status variables created for automatically by NeoDBpro. Replace "ID" with your Database ID and "Table" with the name of your table.

[ID.Table.\$RecNum]	This is the current active record's position from the beginning of the file based on the current sort/query.
[ID.Table.\$RecCount]	If a query is active, the result reflects the number of found records. When no query is active, the result reflects the total number of records in the database.
[ID.Table.\$State]	This is the current state of the table. Possible values include "Browse", "Edit", etc.
[ID.\$Status]	Contains the connection status of the database. This can be either "Connected" or "Not Connected".

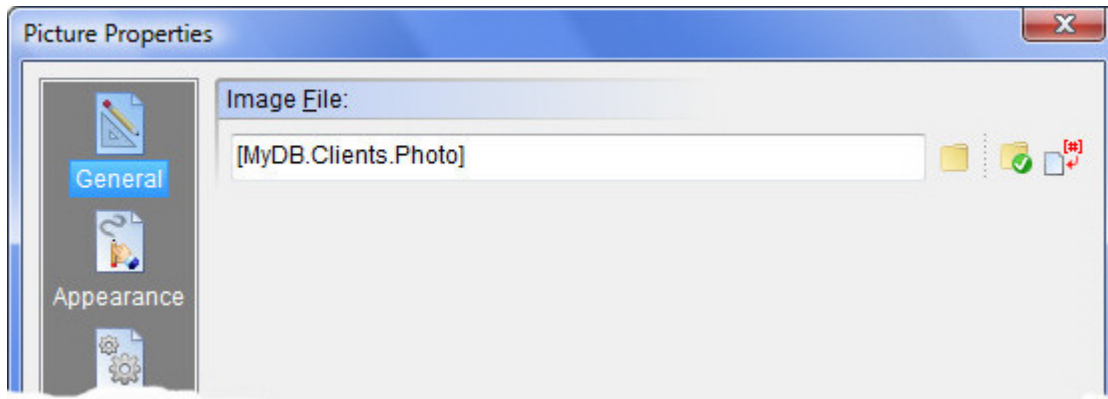
Created with the Standard Edition of HelpNDoc: [Free Kindle producer](#)

Picture Field Considerations

The Picture field type, sometimes called a "binary", "varbinary" or "blob" field, is designed to hold large blocks of non-standard, odd-sized data like photographs, sound, videos, etc. Creating picture-enabled databases with NeoDBpro is fairly straightforward, but this task will be easier once you learn how the picture field works.

Working With Pictures

As you learned in [Working with Data](#), NeoDBpro automatically creates VisualNEO for Windows variables for each field in a table. Picture fields work a little differently. Instead of the picture field variable containing the actual bits that make up the image, the variable contains the name of a temporary file containing the image. To view the image, all you need to do is add the picture field variable (instead of a file name) to one of VisualNEO for Windows's Picture objects. For example, suppose we created a table called Clients that contained a picture field called "Photo". We would link the picture field variable [MyDB.Clients.Photo] to our VisualNEO for Windows Picture object like this:



That's all there is to it. Just like other types of database fields, the contents of the picture object change automatically as you navigate the database.

Adding Images to a Database

You've created a table that contains a picture field and linked it to a picture object, but how do you get images into the database? It's simple, just set the picture field variable to the name of a file that contains the image you want to add to the current record. For example:

```
SetVar "[MyDB.Clients.Photo]" "C:\My Images\Sally.jpg"
```

NeoDBpro will detect the variable change and read the contents of the file into the database. To make it even easier you can use VisualNEO for Windows's FileOpenBox action to select an image file. For example:

```
FileOpenBox "Add Picture" "Images|*.bmp;*.gif;*.jpg;*.tif;*.png;*.pcx" ""
"[FName]" ""
If "[FName]" "<>" ""
    SetVar "[MyDB.Clients.Photo]" "[FName]"
EndIf
```

You can use any VisualNEO for Windows compatible image file (BMP, GIF, JPEG, TIFF, PNG, PCX, WMF, ICO).

You can empty the contents of a picture field by setting the field variable to an empty string. For example:

```
SetVar "[MyDB.Clients.Photo]" ""
```

Limitations of Pictures Fields

Picture fields are stored as binary data, so they cannot be used to perform sorts or queries and cannot be exported or imported using the normal import/export actions. However, you can export individual picture fields one at a time using [dbpExportBlob](#). If you want to create a searchable photo database, you will need to create a string field to store a description or list of keywords describing the contents of the picture field.

Advanced Uses for Picture Fields

Even though NeoDBpro defines its blob field as a picture, you can use it to store just about any type of file. Replace the picture object in the above example with a media player object and you have an audio/video clip database. Replace the picture object with an article object and you have an text clipping database. Basically, any type of file can be stored in a picture field.

Defining Colors

The [dbpSetGridProperties](#) Action allows you to specify colors to customize the appearance of a table grid. NeoBook's interface provides a simple color selector that makes this process easy, but you can also define colors manually if you prefer to type actions into the Action Editor. There are two methods you can use to specify colors in the parameters that require them. These methods are defined below:

Method 1:

Simply enter one of the standard Windows color codes from the list below:

Black	Maroon	Green	Olive
Navy	Purple	Teal	Gray
Silver	Red	Lime	Blue
Fuchsia	Aqua	Yellow	White
Background	ActiveCaption	InactiveCaption	Menu
Window	WindowFrame	MenuText	WindowText
CaptionText	ActiveBorder	InactiveBorder	AppWorkSpace
Highlight	HightlightText	BtnFace	BtnShadow
GrayText	BtnText	InactiveCaptionText	BtnHighlight
3DDkShadow	3DLight	InfoText	InfoBk

The first 16 color codes (Black-White) are generic colors that will appear the same on all computer systems. The other colors are user customizable and can be changed using the Windows Control Panel, so other computers will likely display these colors differently that they appear on your PC. For example:

"FontColor=Purple"

Method 2:

Alternatively, you can specify a custom color using its Red, Green and Blue (RGB) components. Every color that can be displayed on a computer screen is composed of specific quantities of red, green and blue light. By specifying levels (from 0 to 255) of these three colors, you can create any one of the 16 million colors in the computer's palette. In NeoBook, you can specify custom colors using this format:

"FontColor=Red,Green,Blue"

For example, to create pure blue you would specify 0 for red, 0 for blue and 255 for blue:

"FontColor=0,0,255"

For pure green, you would specify 0 for red, 255 for green and 0 for blue:

"FontColor=0,255,0"

You can get really fancy by specifying varying amounts of each of the three values. For example, the following will create a pale shade of blue:

"FontColor=176,232,248"

Created with the Standard Edition of HelpNDoc: [Free iPhone documentation generator](#)

Distributing Database Publications

You've finished building your NeoDBpro database, and you're ready to distribute your

publication - now what? Here are a few things you will need to consider when distributing a VisualNEO for Windows publication that contains a database:

Distributing Database Files

When distributing database-enabled publications there are a couple of things you need to keep in mind. Since VisualNEO for Windows doesn't really understand what a database is, it won't automatically include database files when compiling publications using NeoDBpro. Therefore, you will need to take some special steps to insure that any required files are transported along with your compiled publications. Here are several methods you can use to accomplish this:

1. If your publication will include a setup program, you may be able to add your database files to VisualNEO for Windows's Setup Baggage (found on the Setup page of the Compiler).
2. Use VisualNEO for Windows's ExtractFile action to embed your database files inside the compiled publication exe. ExtractFile must be executed at some point before your publication attempts to open the database. The publication's Startup action or the first page's Enter action are good places to put ExtractFile commands. For Example, to extract an Access mdb file into the folder where the compiled publication is located:

```
ExtractFile "C:\Samples\AddressBook.mdb" "[PubDir]addressbook.mdb"
```

3. Package your database files along with your publication exe inside a zipped or self-extractive archive.
4. Use a professional setup/installation utility designed to work with your database. Some installation utilities include special functions for installing database ODBC drivers, runtime files, etc.
5. Create your database programmatically using NeoDBpro actions and SQL commands. The example below checks to see if the database exists, then creates the database, table and even some data:

```
FileExists "[PubDir]AddressBook.mdb" "[Result]"
If "[Result]" "=" "False"
    .database doesn't exist, so create it from scratch
    dbpCreateAccessDatabase "[PubDir]AddressBook.mdb" "Encrypted=No"
    dbpOpenAccessDatabase "AddrBook" "[PubDir]AddressBook.mdb" ""
    .create table
    dbpCreateTable "AddrBook" "States" "Abbr Char(2);Name String(32)"
    .insert data into table
    dbpExecSQL "AddrBook" "INSERT INTO States VALUES('AL', 'Alabama')" ""
    dbpExecSQL "AddrBook" "INSERT INTO States VALUES('AK', 'Alaska')" ""
    dbpExecSQL "AddrBook" "INSERT INTO States VALUES('AZ', 'Arizona')" ""
    dbpExecSQL "AddrBook" "INSERT INTO States VALUES('AR', 'Arkansas')" ""
    ...
EndIf
```

The methods outlined here work best with databases, like Access, that package everything into a single, easy-to-distribute file. Other databases like MySQL and Oracle, designed for use on network servers, require special installations and are not easily moved between computers. Some databases also require special distribution licenses, so it always a good idea to consult your database documentation to determine the best method for distributing your database.

Regardless of the method you use to distribute your database files, you will want to insure that NeoDBpro can find them. For file-based databases, like Access, the simplest method is to place your database files in the same folder as your publication exe and use VisualNEO for Windows's global [PubDir] variable as part of the file name. For example:

```
dbpOpenAccessDatabase "AddrBook" "[PubDir]AddressBook.mdb" ""
```

For server-based databases like MySQL and Oracle, the key to a smooth installation is identifying the correct connection string. Server names, login names, passwords, etc. will need to be customized for each installation. In multi user settings you may want to include a login screen or a configuration option to allow for these differences.

Other Important Files

Many types of databases will consist of more than one file. dBase databases that contain memo or blob fields will also generate a companion dbt file. For example, if your database file is called "sample.dbf", then its companion memo file would be called "sample.dbt". In order for the database to remain intact, both of these files must be distributed along with your publication.

See also [Font Considerations](#).

Created with the Standard Edition of HelpNDoc: [iPhone web sites made easy](#)

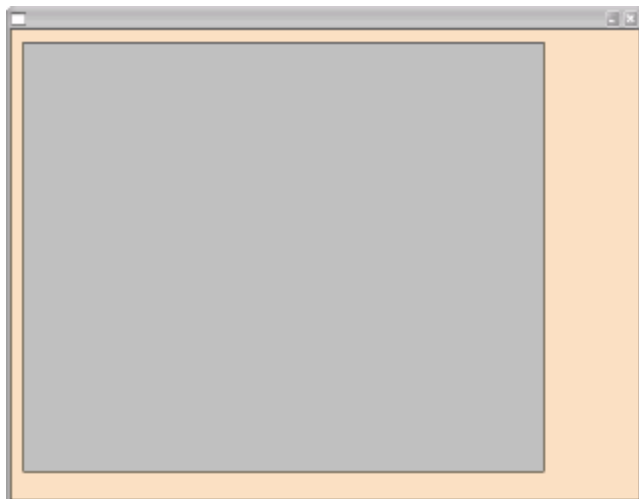
NeoDBpro Tutorial

Creating powerful database applications is relatively simple because NeoDBpro insulates you from much of the complexities of SQL. The steps below demonstrate how to open and display an existing MS Access database with NeoDBpro:

Getting Started

We recommend that you start with a new VisualNEO for Windows publication.

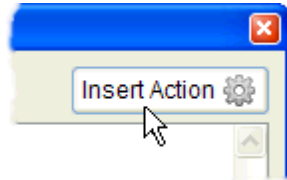
1. First you will need to use VisualNEO for Windows's **Rectangle** tool  to define the location on the screen where you want the database's table [grid](#) to appear. To do this, simply select the Rectangle tool from VisualNEO for Windows's Tool Palette and draw a box on your publication. When your publication runs, the rectangle will be replaced by the table grid, so make sure that it is fairly large. After creating the rectangle, your screen should look like this:



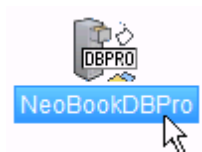
- 2.** Create a **Push Button**  next to the rectangle.
- 3.** Type Open in the Push Button's **Caption** property.
- 4.** Click the **Actions** icon on the left side of the Push Button Properties dialog.



5. Click the **Insert Action** button.



6. Locate and select the **NeoDBpro** category among the icons on the left side of the Action Selector. If you don't see an icon for NeoDBpro, then you have not properly installed the plug-in. See **Install Plug-Ins** in VisualNEO for Windows's Options menu.



7. After selecting the NeoDBpro icon, select [dbpOpenAccessDatabase](#) from the right side of the screen. The following dialog box will appear:

dbpOpenAccessDatabase Properties

Database ID: (required)
 MyFirstDB The ID is simply a name that you will use to refer to this database after it's open.

Database File Name: (MS Access mdb or accdb files only)
 [PubDir]AddressBook.mdb

Password: (if required)
 [Empty field]

Options:
 Cursor Location: Client

dbpOpenAccessDatabase
 Open a Microsoft Access (MDB, ACCDB) database.

NeoBookDBPro™
 ©2007-2011 NeoSoft Corp.

OK Cancel Help

8. The first parameter is the **Database ID**, which is simply a name that we will use to refer to this database in the future. It can be anything you like, but it's often easiest to use something short and descriptive. We'll use the ID later for other database commands.

For this tutorial, enter MyFirstDB in the Database ID field.

9. Next, type the name of the MS Access (*.mdb or *.accdb) file you want to display in the **Database File Name** field. If you don't know the name of the file you can click the small folder icon to the right of the field to select from a list. For this tutorial, we will use the sample AddressBook.mdb file included with this plug-in. You can find the AddressBook database in the following folder:

```
..\Documents\VisualNEO for Windows\NeoDBpro  
Samples\AddressBook\AddressBook.mdb
```

Enter this into the Database File Name field. If you installed VisualNEO for Windows or the plug-in in a different folder, then change the path above to reflect the location you selected.

Note: Since VisualNEO for Windows can't compile a database file inside your publication exe, you will need to take steps to insure that the file is included along with your finished project. The easiest way to do this is to make sure that the database is always placed in the same folder as your compiled publication, and replace the drive and path in the Database File Name field with VisualNEO for Windows's [PubDir] variable. For example, to open a database file named AddressBook.mdb, change the Database File Name field to:

```
[PubDir]AddressBook.mdb
```

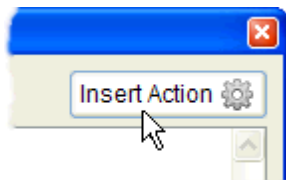
At runtime, VisualNEO for Windows will automatically replace [PubDir] with the location of your compiled publication which should be the same as the location of your database file.

To insure that your publication also works when testing in VisualNEO for Windows, you can place a copy of your database file in the same folder as your uncompiled pub file.

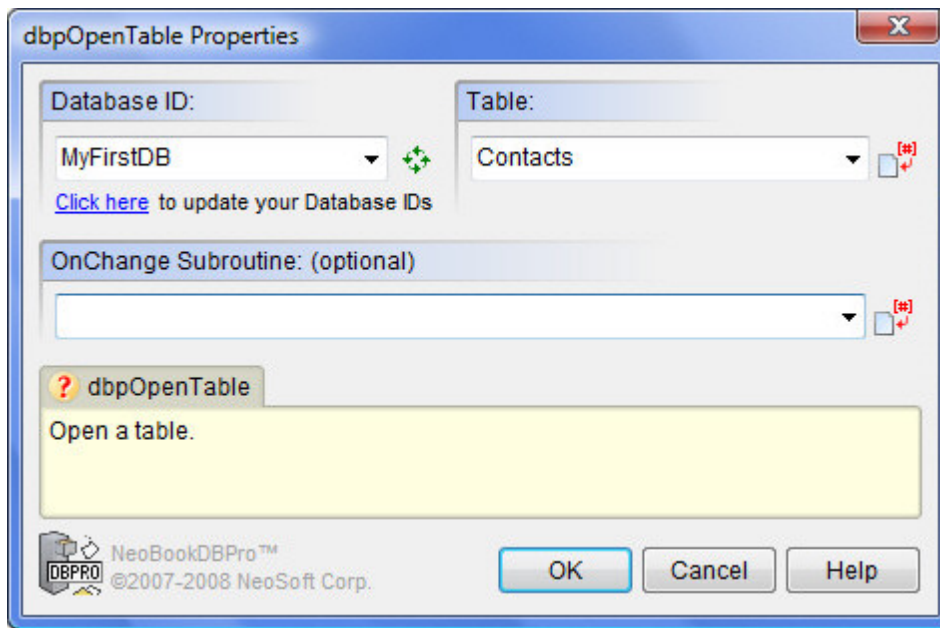
10. If the database you selected is password protected, enter your password in the **Password** field. Otherwise, leave this field blank.

11. Click **OK** to assign the dbpOpenAccessDatabase action to the button.

12. Click the **Insert Action** button again.



13. Locate and select the [dbpOpenTable](#) action from the right side of the screen. The following dialog box will appear:



14. Enter MyFirstDB in the **Database ID** field.

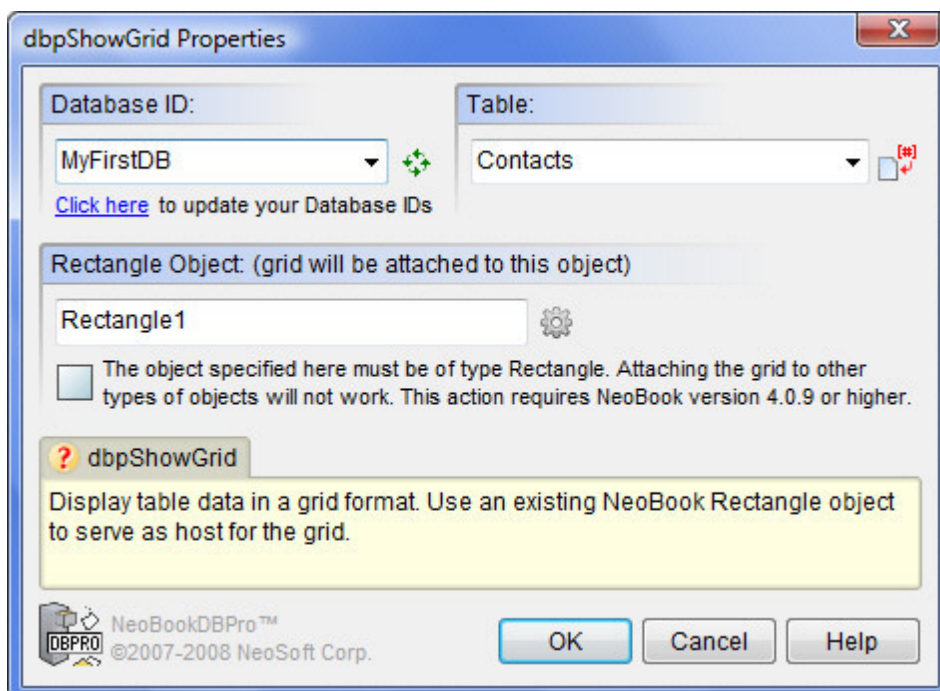
15. If you defined the database correctly in step 9 above, you should be able to select Contacts from **Table** field combo box. If the table list is empty, then the Database File Name is probably incorrect. Go back to step 9 and correct the problem.

16. Click **OK** to assign the dbpOpenTable action to the button.

Note: After clicking OK, you may be asked: "Would you like to create text entry fields and labels for use with this table?" This is a very handy feature that you will find useful in the future, but for this tutorial you should answer "No" to this question.

17. Click the **Insert Action** button again.

18. Locate and select the [dbpShowGrid](#) action from the right side of the screen. The following dialog box will appear:



19. Enter MyFirstDB in the **Database ID** field.

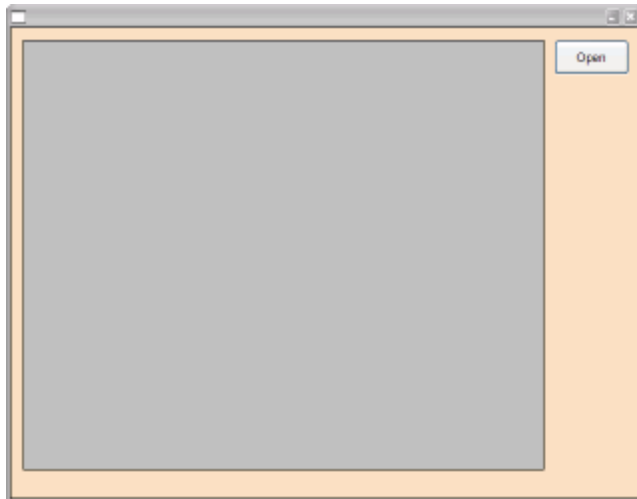
20. Enter Contacts in the **Table** field.

21. In the **Rectangle Object** field, enter the name of the Rectangle object you created in step 1. You can also click the small gear icon to the right of the field and select the rectangle from a list.

22. Click **OK** to assign the pdfShowGridf action to the button. You should now have the following actions assigned to your Push Button:

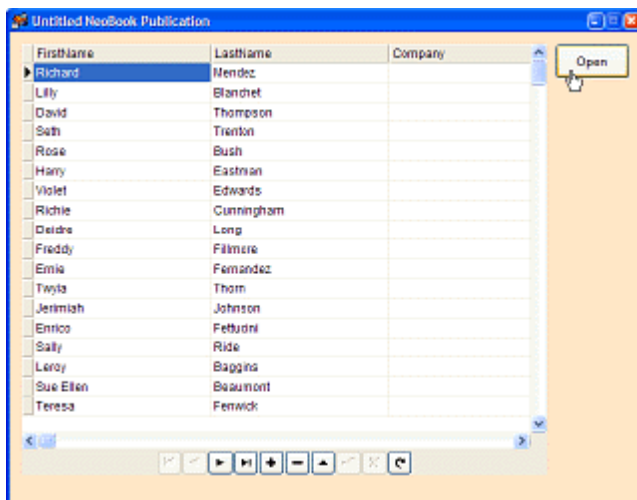
```
dbpOpenAccessDatabase "MyFirstDB" "C:\Program Files\VisualNEO for
Windows\PlugIns\NeoDBpro\Sample Pubs\AddressBook\AddressBook.mdb" ""
dbpOpenTable "MyFirstDB" "Contacts" ""
dbpShowGrid "MyFirstDB" "Contacts" "Rectangle1"
```

23. Click **OK** again to save the Push Button. Your screen should now look like this:



24. Let's test the publication. Select **Run** or **Run (From Start)** from the VisualNEO for Windows's **Book** menu.

25. Click the **Open** button to load and display the Contacts table. Your screen should look like this:



26. When you're finished testing, click the publication window's **Close** button or press the

Esc key on your keyboard.

To compile this publication, simply select **Compile** from VisualNEO for Windows's **Book** menu and compile as you would any other project. Then make sure to distribute both the compiled pub and the AddressBook.mdb file. See [Distributing Database Publications](#) for more information on this topic.

Created with the Standard Edition of HelpNDoc: [Free iPhone documentation generator](#)

Action Command Reference

This section contains descriptions of each of the Action commands found in NeoDBpro. After installing NeoDBpro, the Actions below should automatically appear in VisualNEO for Windows's **Insert Action** list in the NeoDBpro section. If you're not familiar with the basic concept of using Actions in VisualNEO for Windows, please read "Understanding Actions and Variables" in your VisualNEO for Windows Help file.

Categorical list of NeoDBpro Action commands:

Database

[dbpOpenDatabase](#)
[dbpOpenAccessDatabase](#)
[dbpCreateAccessDatabase](#)
[dbpCompactAccessDatabase](#)
[dbpCloseDatabase](#)

Tables

[dbpOpenTable](#)
[dbpCreateTable](#)
[dbpCreateView](#)
[dbpDropTable](#)
[dbpCloseTable](#)
[dbpGetTableNames](#)
[dbpGetFieldNames](#)
[dbpGetFieldDefs](#)
[dbpAddField](#)
[dbpDropField](#)
[dbpTableExists](#)
[dbpFieldExists](#)
[dbpDefineRelationship](#)
[dbpEndRelationship](#)

Grids

[dbpShowGrid](#)
[dbpHideGrid](#)
[dbpSetGridProperties](#)
[dbpSetGridBackground](#)
[dbpSetFieldProperties](#)
[dbpGetActiveField](#)
[dbpSetColumnTitles](#)
[dbpSetColumnOrder](#)
[dbpSetColumnWidths](#)
[dbpGetColumnWidths](#)
[dbpSetRowHeight](#)
[dbpGetRowHeight](#)
[dbpGetGridClientInfo](#)
[dbpDefineValueList](#)

Navigation

[dbpNext](#)
[dbpPrev](#)
[dbpFirst](#)
[dbpLast](#)
[dbpGotoRecord](#)

Search/Query

[dbpQuery](#)
[dbpShowAll](#)
[dbpFind](#)
[dbpFindNext](#)

Advanced SQL

[dbpExecSQL](#)

Sort/Index

[dbpSort](#)
[dbpUnSort](#)
[dbpGetIndexNames](#)
[dbpCreateIndex](#)
[dbpDropIndex](#)

Add/Modify

[dbpAddRecord](#)
[dbpDeleteRecord](#)
[dbpDeleteAll](#)
[dbpEmptyTable](#)
[dbpStrReplace](#)
[dbpRefresh](#)

Advanced Add/Modify

[dbpSetAutoEdit](#)
[dbpSaveEdits](#)
[dbpCancelEdits](#)

Simple Reporting

[dbpListPrint](#)
[dbpListAddHeader](#)
[dbpListAddFooter](#)
[dbpListClearHeadersAndFooters](#)

Advanced Reporting

[dbpPrintReport](#)
[dbpPreviewReport](#)
[dbpSetPreviewHints](#)

Import/Export

[dbpImportFromCSV](#)
[dbpExportToCSV](#)
[dbpExportToHTML](#)
[dbpExportToXML](#)
[dbpExportBlob](#)
[dbpFieldToVar](#)
[dbpRecordToVar](#)
[dbpVarToRecord](#)

Stored Procedures

[dbpGetProcedureNames](#)
[dbpGetProcedureParameters](#)
[dbpSetParameter](#)
[dbpAddParameter](#)
[dbpClearParameters](#)
[dbpExecProc](#)

Math

[dbpSum](#)
[dbpMin](#)
[dbpMax](#)
[dbpAvg](#)

Miscellaneous

[dbpPopupValueList](#)[dbpShowErrors](#)[dbpTranslateHints](#)[dbpPopupDateSelector](#)

Created with the Standard Edition of HelpNDoc: [Write EPub books for the iPad](#)

Database

dbpOpenDatabase

Purpose:	Open a database using a file name or server connection string. A connection string should include a provider, data source, user name, password, etc. as required by server. See Opening a Database for more information.
Category:	Database
Syntax:	dbpOpenDatabase "database id" "file name or connection string" <i>database id</i> A name that you want to use to refer to this database in the future. <i>file name or connection string</i> The type of database you're using dictates whether you can use a simple file name or connection string here. See Opening a Database for more information.
Example:	The following example opens a dBase format file called Sample.dbf: <pre>dbpOpenDatabase "DB1" "[PubDir]Sample.dbf"</pre> Here we use a connection string to open a MySQL database: <pre>dbpOpenDatabase "MySQL" "Provider=MSDASQL.1;Driver={MySQL ODBC 3.51 Driver};Server=localhost;Database=test;User=root;Password=apple;Option=3"</pre> Here is a connection string to open a SQLite database: <pre>dbpOpenDatabase "SQLite" "Provider=MSDASQL.1;Driver=SQLite3 ODBC Driver;Database=C:\Program Files\SQLite ODBC Driver\test.db"</pre> Note: SQLite requires that you have a compatible ODBC driver installed on your PC. If you don't have a SQLite ODBC driver, you can find one at the following web site: http://www.ch

	werner.de/sqliteodbc/
SQL Equivalent:	USE <i>database</i>

dbpOpenAccessDatabase

Purpose:	Open a Microsoft Access (MDB or ACCDB) format database file. To open other types of databases use dbpOpenDatabase instead. See Opening a Database for more information.								
Category:	Database								
Syntax:	dbpOpenAccessDatabase "database id" "file name" "options" <i>database id</i> A name that you want to use to refer to this database in the future. <i>file name</i> The database file name. <i>options</i> This is a compound parameter and can contain any combination of the following items: <table border="0"> <tr> <td>Password=<i>pwd</i></td><td>Replace <i>pwd</i> with the password required to open this database. Leave this option blank if the database is not password protected.</td></tr> <tr> <td>CursorLocation=<i>value</i></td><td>This defines the method used to access data. Value may be one of the following:</td></tr> </table> <table border="0"> <tr> <td>Client</td><td>Client-side processing. (Recommended for small and medium-sized databases.)</td></tr> <tr> <td>Server</td><td>Server-side processing. (Recommended for advanced users working with very large databases.)</td></tr> </table> By default NeoDBpro uses something called a client-side cursor. This is a method of accessing data that relies on the client application to handle most of the data storage and processing. Client-side processing is very flexible and generally provides the fastest performance for small to medium sized databases. For large databases, however, a client-side cursor can sometimes consume too many system resources resulting in poor performance. To compensate for this problem, you may want to use a server-side cursor when working with very large databases. With a client-side cursor (the default) all data is copied to the local machine and processed there. This provides access to features not normally supported by servers such as sorting and indexing. When using a SQL query, only the data returned is copied to the local machine. A server-side cursor doesn't provide as much flexibility, but is often more appropriate for large databases, and may be required when the size of a database exceeds the available memory and disk space available on a local machine. Separate multiple items in a compound variable with semicolons (;). This parameter may be left blank if none of the options are required.	Password= <i>pwd</i>	Replace <i>pwd</i> with the password required to open this database. Leave this option blank if the database is not password protected.	CursorLocation= <i>value</i>	This defines the method used to access data. Value may be one of the following:	Client	Client-side processing. (Recommended for small and medium-sized databases.)	Server	Server-side processing. (Recommended for advanced users working with very large databases.)
Password= <i>pwd</i>	Replace <i>pwd</i> with the password required to open this database. Leave this option blank if the database is not password protected.								
CursorLocation= <i>value</i>	This defines the method used to access data. Value may be one of the following:								
Client	Client-side processing. (Recommended for small and medium-sized databases.)								
Server	Server-side processing. (Recommended for advanced users working with very large databases.)								
Example:	dbpOpenAccessDatabase "DB1" "[PubDir]AddressBook.mdb" "password=kitty123"								

SQL Equivalent:	N/A
-----------------	-----

dbpCreateAccessDatabase

Purpose:	Create a new Microsoft Access (MDB or ACCDB) format database file. This action only creates the database - it does not open it. You must use dbpOpenAccessDatabase to open the database for editing.
Category:	Database
Syntax:	dbpCreateAccessDatabase "file name" "options" <i>file name</i> The database file name. <i>options</i> This is a compound parameter and can contain any combination of the following items: Password=<i>pwd</i> Replace <i>pwd</i> with the password you wish to use to protect this database. The password assigned here must be passed to dbpOpenAccessDatabase in order to open the database in the future. Please use caution when assigning passwords. If you forget the password, you will not be able to open the database. Encrypted=Yes/No Use "Encrypted=Yes" to encrypt the database file to protect the data from being viewed with a hex editor. Separate multiple items in a compound variable with semicolons (;). This parameter may be left blank if a password and encryption are not desired.
Example:	dbpCreateAccessDatabase "[PubDir]AddressBook.mdb" "Password=kitty123;Encrypted=Yes" To create unprotected, unencrypted database you can simply leave the options parameter blank: dbpCreateAccessDatabase "[PubDir]AddressBook.mdb" ""
SQL Equivalent:	N/A

dbpCompactAccessDatabase

Purpose:	Compress a Microsoft Access (MDB or ACCDB) format database file and remove space occupied by deleted records. When records are deleted using dbpDeleteRecord , the physical size of the database file does not necessarily decrease. Instead the database may simply mark the record as deleted, but the space occupied by the record remains in the file. dbpCompactAccessDatabase removes any records marked as deleted and reduces the size of the database file.
Category:	Database
Syntax:	dbpCompactAccessDatabase "file name" "password" <i>file name</i> The database file name. <i>options</i> This is a compound parameter and can contain any combination of the following items: Password=<i>pwd</i> Replace <i>pwd</i> with the password required to access the database.

	Leave blank if no password is required. Encrypted=Yes/No Use "Encrypted=Yes" to encrypt the database file while compressing the data. Separate multiple items in a compound variable with semicolons (;). This parameter may be left blank if a password and encryption are not required.
Example:	dbpCompactAccessDatabase "[PubDir]AddressBook.mdb" ""
SQL Equivalent:	N/A

dbpCloseDatabase

Purpose:	Close a database previously opened with dbpOpenDatabase or dbpOpenAccessDatabase . Calling dbpCloseDatabase is optional since any databases or tables that are open when a publication is shutdown will automatically be closed.
Category:	Database
Syntax:	dbpCloseDatabase "database id" <i>database id</i> The name assigned to the database you want to close.
Example:	dbpCloseDatabase "DB1"
SQL Equivalent:	N/A

Created with the Standard Edition of HelpNDoc: [Easily create EBooks](#)

Tables

dbpOpenTable

Purpose:	Open a database table. Optionally, you can also specify a script from your publication's Subroutine Action that you want executed whenever the table changes.
Category:	Tables
Syntax:	dbpOpenTable "database id" "table" "subroutine" <i>database id</i> The name assigned to the database containing the table you want to open. <i>table</i> The name of the table you wish to open. <i>subroutine</i> The name of a subroutine from your publication's Subroutine Action. The subroutine specified here will be automatically executed whenever the table is updated or the current record number changes. This can be useful if you want an activity to take place whenever the reader displays a different record. Subroutines are entered from the Actions page of VisualNEO for Windows's Book Properties screen. VisualNEO for Windows's help file contains more information on using subroutines. Note: By default, dbpOpenTable will perform a "show all" type query on the table as soon as it's opened. This is desirable for small and medium sized tables, but may degrade performance with extremely large tables. Instead, you may prefer to open large tables with the dbpExecSQL action, so you can limit the amount of data returned and improve performance.

Example:	dbpOpenTable "DB1" "sales" "OnChangeSub"
SQL Equivalent:	SELECT * FROM <i>table</i>

dbpCreateTable

Purpose:	Create a new database table.										
Category:	Tables										
Syntax:	dbpCreateTable "database id" "table" "field defs" <i>database id</i> The name assigned to the database where the new table will be created. <i>table</i> The name of the table you wish to create. Attempting to use a name that belongs to an existing table will generate an error. <i>field defs</i> This is a list that defines the types of fields that will comprise each record in the new table. Each field definition consists of a unique field name, the field type and, depending on the field and database type, size and options. Multiple fields are separated by semicolons (;). For example: "field1 type(size) options;field2 type(size) options;..." Field names can be just about anything you like as long as each name is used only once in the same table. Typically a field's name should reflect the type of data it will contain, such as LastName, City, State, etc. Tables store different types of data in specific types of fields. Text is stored in either a string, char or memo type field, while numbers are stored in an integer or float type field. Since VisualNEO for Windows doesn't really distinguish between text and numbers the way traditional programming environments do, you can get by with defining all of your fields as strings if you like. The only exception would be if you plan on loading your database file into another program. In that case, you may want to define your fields using the field types that more closely match your data. Note: Unfortunately, different databases offer different, sometimes incompatible, array of choices when it comes to field or data types. Sometimes even field types with the same names can have different capacities or properties in another database. To be sure you're getting what you expect, check the database's documentation. The types of database fields supported by NeoDBpro are listed below: <table> <tr> <td>String</td><td>Variable-length character data with a maximum of 8,000* characters. Used for storing text or combinations of text and numbers, such as addresses. Can also be used for numbers that do not require calculations, such as telephone numbers, part numbers, postal codes, etc. Some types of databases refer to this field type as a "VarChar". The desired size of the field is specified in parenthesis. For example: String(500)</td></tr> <tr> <td></td><td>Note: For Microsoft Access databases the size of a String field is limited to 255 characters.</td></tr> <tr> <td>Char</td><td>Fixed-length character data with a maximum length of 255* characters. The desired size is specified in parenthesis. For example: Char(35)</td></tr> <tr> <td>Memo</td><td>Variable-length character data with a maximum length of 64,000* characters. This field type is sometimes called a "Text" field.</td></tr> <tr> <td>Boolean</td><td>Can contain either "true" or "false". This field type is sometimes called a Bit.</td></tr> </table>	String	Variable-length character data with a maximum of 8,000* characters. Used for storing text or combinations of text and numbers, such as addresses. Can also be used for numbers that do not require calculations, such as telephone numbers, part numbers, postal codes, etc. Some types of databases refer to this field type as a "VarChar". The desired size of the field is specified in parenthesis. For example: String(500)		Note: For Microsoft Access databases the size of a String field is limited to 255 characters.	Char	Fixed-length character data with a maximum length of 255* characters. The desired size is specified in parenthesis. For example: Char(35)	Memo	Variable-length character data with a maximum length of 64,000* characters. This field type is sometimes called a "Text" field.	Boolean	Can contain either "true" or "false". This field type is sometimes called a Bit.
String	Variable-length character data with a maximum of 8,000* characters. Used for storing text or combinations of text and numbers, such as addresses. Can also be used for numbers that do not require calculations, such as telephone numbers, part numbers, postal codes, etc. Some types of databases refer to this field type as a "VarChar". The desired size of the field is specified in parenthesis. For example: String(500)										
	Note: For Microsoft Access databases the size of a String field is limited to 255 characters.										
Char	Fixed-length character data with a maximum length of 255* characters. The desired size is specified in parenthesis. For example: Char(35)										
Memo	Variable-length character data with a maximum length of 64,000* characters. This field type is sometimes called a "Text" field.										
Boolean	Can contain either "true" or "false". This field type is sometimes called a Bit.										

Integer	A whole number between -32,768 and 32,767*.
BigInt	A very large whole number between -2,147,483,648 and 2,147,483,647*.
AutoInc	A unique sequential number automatically inserted when a record is added. The number will be incremented by one for each new record.
Currency	Monetary values. Accurate to 15 digits to the left of the decimal point and 4 digits to the right. This field type is sometimes called a "Money" field.
Float	Decimal number between 10^{28-1} and 10^{28-1} *.
Date	Date. This type is not supported by MS Access, use DateTime instead.**
Time	Time. This type is not supported by MS Access, use DateTime instead.**
DateTime	Date and time data.**
Picture	Can be used to store just about any type of binary data, including pictures, document files, etc. This field type is sometimes called "binary", "varbinary" and "blob".

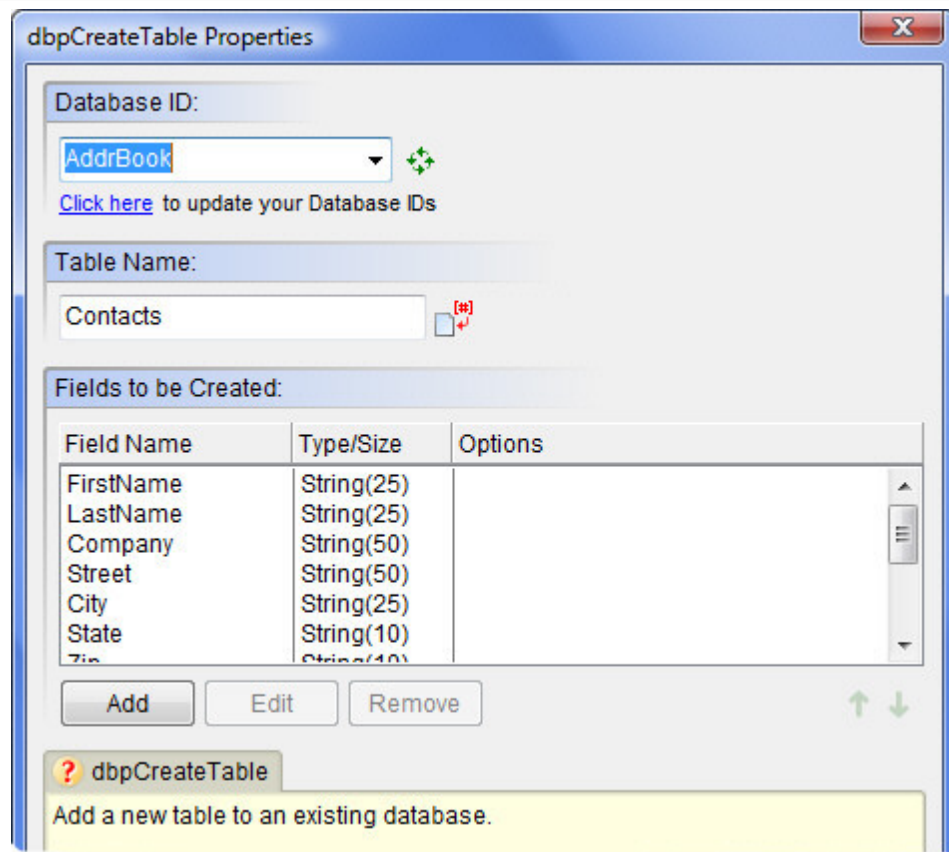
Not all types of databases support this type of field, and there are no standard formats for storing data. This means that data stored here may not be easily exchanged between different types of databases. See [Picture Field Considerations](#) for more information.

*Maximum field sizes vary widely between databases and even among versions of the same database. Check your database's documentation for exact field properties.

Microsoft Access does not distinguish between date and time fields internally. In order to display date and time values correctly, you must manually define the field's appearance using the **DisplayFormat property of the [dbpSetFieldProperties](#) action.

Finally, the field size is used to define the maximum length for String and Char types. Non MS Access databases can optionally specify sizes for Integer, BigInt and Float types. Size is ignored for all other field types. You can reduce the size of your database by specifying maximum sizes for each String and Char fields. For example, a zip code field rarely requires more than 5 characters and last names can usually fit in 25 characters. Allocating more space for these types of fields will make your database file larger than it needs to be.

Because dbpCreateTable can be complex, the easiest way to properly construct this syntax is to select the action from VisualNEO for Windows's Select and Action list and use the dbpCreateTable wizard pictured below:



dbpCreateTable Properties

Database ID: AddrBook

Click here to update your Database IDs

Table Name: Contacts

Fields to be Created:

Field Name	Type/Size	Options
FirstName	String(25)	
LastName	String(25)	
Company	String(50)	
Street	String(50)	
City	String(25)	
State	String(10)	
Zip	String(10)	

Add Edit Remove

? dbpCreateTable

Add a new table to an existing database.

After you define your table using the wizard you will be asked if you want NeoDBpro to create basic Text Entry boxes for each of your table's fields. If you answer "Yes" the Text Entry boxes will be placed onto the Windows clipboard. You can then return to VisualNEO for Windows and select Paste from Edit menu to place the objects onto your publication.

Note: Advanced users may choose to create tables by passing SQL commands to the [dbpExecSQL](#) action.

Example: dbpCreateTable "AddrBook" "Contacts" "FirstName String(25);LastName String(25);Company String(50);Street String(50);City String(25);State String(10);Zip String(10);Country String(50) Default=USA;Telephone String(20);EMail String(35);WebSite String(35);Comments Memo"

SQL Equivalent: CREATE TABLE table (field defs)

dbpCreateView

Purpose: Create an alternate, virtual view of a table. The data displayed in the view is identical to the data in the source table. Only one copy of the data exists in the source table. Edits applied to the view are automatically reflected in the source table and vice versa. Once created, the view is added to the database's list of tables and acts and behaves exactly like a normal table. A view can be opened with [dbpOpenTable](#), displayed with [dbpShowGrid](#) and deleted with [dbpDropTable](#). Deleting a view with dbpDropTable does not affect any of the data which is stored in the source table.

Category: Tables

Syntax: dbpCreateView "database id" "source table" "view name"

database id
The name assigned to the database.

	<i>source table</i> The name of the table to be used as the source for the view. <i>view name</i> The name to use for the view. This must be a unique name not used by any other tables or views in the database.
Example:	dbpCreateView "AddrBook" "Clients" "ViewOfClients"
SQL Equivalent:	CREATE VIEW <i>view name</i> AS SELECT * FROM <i>source table</i>

dbpDropTable

Purpose:	Delete a database table. Use this action with caution. Once a table is deleted it cannot be recovered.
Category:	Tables
Syntax:	dbpDropTable "database id" "table" <i>database id</i> The name assigned to the database containing the table you want to delete. <i>table</i> The name of the table to delete.
Example:	dbpDropTable "DB1" "sales"
SQL Equivalent:	DROP TABLE <i>table</i>

dbpCloseTable

Purpose:	Close a database table. Calling dbpCloseTable is optional since any tables that are open when a publication is shutdown will automatically be closed.
Category:	Tables
Syntax:	dbpCloseTable "database id" "table" <i>database id</i> The name assigned to the database containing the table you want to close. <i>table</i> The name of the table to Close.
Example:	dbpCloseTable "DB1" "sales"
SQL Equivalent:	N/A

dbpGetTableNames

Purpose:	Obtain the names of all tables in a database and store the results in a variable. Table names will be separated by the specified delimiter.
Category:	Tables
Syntax:	dbpGetTableNames "database id" "delimiter" "variable" <i>database id</i> The name assigned to the database. <i>delimiter</i>

	The character or characters used to separate the table names. <i>variable</i> The name of the variable where the table names will be stored. Each table name will be separated by the specified delimiter.
Example:	dbpGetTableNames "AddrBook" "[#13]" "[TableList]"
SQL Equivalent:	SHOW TABLES

dbpGetFieldNames

Purpose:	Obtain the names of all field names in a table and store the results in a variable. Field names will be separated by the specified delimiter.
Category:	Tables
Syntax:	dbpGetFieldNames "database id" "table" "delimiter" "variable" <i>database id</i> The name assigned to the database containing the table. <i>table</i> The name of the table whose fields you want to retrieve. <i>delimiter</i> The character or characters used to separate the field names. <i>variable</i> The name of the variable where the field names will be stored. Each field name will be separated by the specified delimiter.
Example:	dbpGetfieldNames "AddrBook" "Contacts" "[#13]" "[FieldList]"
SQL Equivalent:	DESCRIBE <i>table</i>

dbpGetFieldDefs

Purpose:	Obtain the field definitions used to create the table. Each field definition consists of a unique field name, the field type and, depending on the field and database type, size and options. Multiple fields will be separated by the specified delimiter. For example: field1 type(size) options;field2 type(size) options;... This format is also used by dbpCreateTable. Note: Different databases offer different, sometimes incompatible, array of choices when it comes to field types. Sometimes even field types with the same names can have different capacities or properties in another database. For this reason, the field defs returned by dbpGetFieldDefs may differ slightly from the ones used when the table was created.
Category:	Tables
Syntax:	dbpGetFieldDefs "database id" "table" "delimiter" "variable" <i>database id</i> The name assigned to the database containing the table. <i>table</i> The name of the table whose field definitions you want to retrieve. <i>delimiter</i> The character or characters used to separate the field definitions.

	<i>variable</i> The name of the variable where the field definitions will be stored. Each field definition will be separated by the specified delimiter.
Example:	dbpGetfieldDefs "AddrBook" "Contacts" "[#13]" "[FieldInfo]"
SQL Equivalent:	DESCRIBE <i>table</i>

dbpAddField

Purpose:	Add a field to an existing table.
Category:	Tables
Syntax:	dbpAddField "database id" "table" "field" "field defs" <i>database id</i> The name assigned to the database containing the table. <i>table</i> The name of the table where the new field will be added. <i>field</i> The name of the new field. <i>field defs</i> The field's definition. A field definition consists of a unique field name, the field type and, depending on the field and database type, size and options. See dbpCreateTable for more information.
Example:	dbpAddfield "AddrBook" "Contacts" "Occupation" "String(50)"
SQL Equivalent:	ALTER TABLE <i>table</i> ADD COLUMN <i>field name</i> <i>field defs</i>

dbpDropField

Purpose:	Remove a field and all associated data from a table.
Category:	Tables
Syntax:	dbpDropField "database id" "table" "field" <i>database id</i> The name assigned to the database containing the table. <i>table</i> The name of the table containing the field to be deleted. <i>field</i> The name of the field to delete.
Example:	dbpDropfield "AddrBook" "Contacts" "Occupation"
SQL Equivalent:	ALTER TABLE <i>table</i> DROP COLUMN <i>field</i>

dbpTableExists

Purpose:	Determine if a database contains a specific table.
Category:	Tables

Syntax:	<code>dbpTableExists "database id" "table" "variable"</code> <i>database id</i> The name assigned to the database containing the table. <i>table</i> The name of the table. <i>variable</i> The name of the variable where the result will be stored. The variable will be set to "True" if the table exists or "False" if it does not.
Example:	<pre>dbpTableExists "AddrBook" "Contacts" "[Result]" If "[Result]" "=" "True" MsgBox "Hello" "A table named Contacts already exists." EndIf</pre>
SQL Equivalent:	N/A

dbpFieldExists

Purpose:	Determine if a table contains a specific field.
Category:	Tables
Syntax:	<code>dbpFieldExists "database id" "table" "field" "variable"</code> <i>database id</i> The name assigned to the database containing the table. <i>table</i> The name of the table containing the field. <i>field</i> The name of the field. <i>variable</i> The name of the variable where the result will be stored. The variable will be set to "True" if the field exists or "False" if it does not.
Example:	<pre>dbpFieldExists "AddrBook" "Contacts" "LastName" "[Result]" If "[Result]" "=" "True" MsgBox "Hello" "A field named LastName already exists." EndIf</pre>
SQL Equivalent:	N/A

dbpDefineRelationship

Purpose:	Establish a master-detail relationship between two tables. Once a relationship is defined, navigational changes to the master table will automatically display matching records in the detail table. In order to link two tables, they must share a common field such as a customer name, order number or part number. The contents of the master table's master field will be used to query the linked field in the detail table.
Category:	Tables
Syntax:	<code>dbpDefineRelationship "database id" "mastertable" "masterfield" "detailtable" "detailfield" "fieldmask"</code> <i>database id</i> The name assigned to the database containing the tables to be linked.

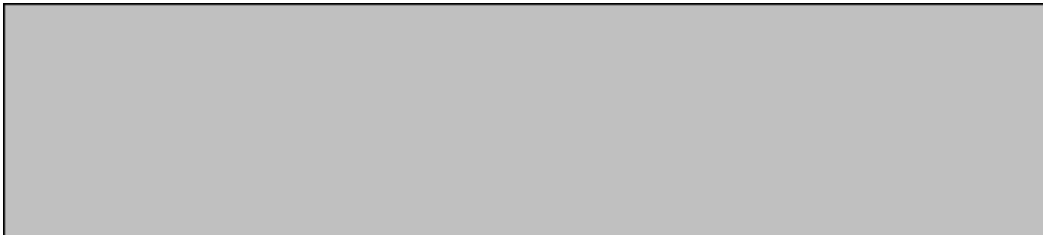
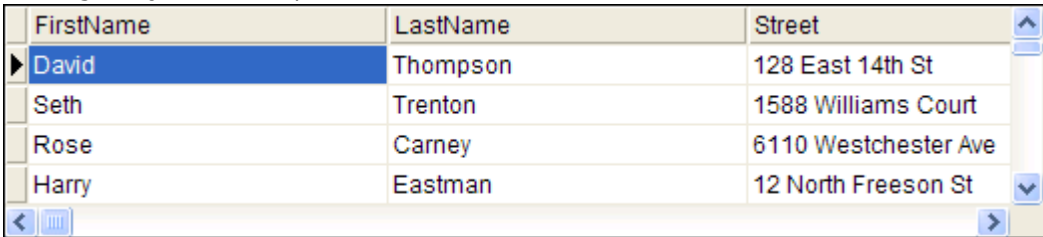
	<i>mastertable</i> The name of the master table. <i>masterfield</i> The name of the field in the master table to link. <i>detailtable</i> The name of the detail table. <i>detailfield</i> The name of the field in the detail table to link. <i>fieldmask</i> A list of fields (separated with commas) from the detail table to be included in the query that controls the master-detail relationship. The fields listed here will displayed in the detail table's grid. Fields not listed here will be omitted from the grid. (Basically, you should be able to use anything that can be inserted between SELECT and FROM in SQL.) To use all fields, leave this parameter blank. Note: The master field and detail field must be of the same or compatible types.
Example:	<pre>dbpDefineRelationship "DB1" "Orders" "orderNum" "OrderDetails" "orderNum" ""</pre> The fieldmask parameter can be used to limit which fields are displayed. For example: <pre>dbpDefineRelationship "DB1" "Orders" "orderNum" "OrderDetails" "orderNum" "Quantity, Description, UnitPrice"</pre> You can even use the fieldmask to add a temporary calculated field to the table. For example: <pre>dbpDefineRelationship "DB1" "Orders" "orderNum" "OrderDetails" "orderNum" "Quantity, Description, UnitPrice, Quantity * UnitPrice AS Total"</pre>
SQL Equivalent:	See JOIN or INNER JOIN.

dbpEndRelationship

Purpose:	Clear a master-detail relationship previously established with dbpDefineRelationship . Calling dbpEndRelationship is optional since relationships are automatically terminated when a table or database is closed. If more than one relationship exists between two table this action will clear all of them.
Category:	Tables
Syntax:	<pre>dbpEndRelationship "database id" "mastertable" "detailtable"</pre> <i>database id</i> The name assigned to the database containing the linked tables. <i>mastertable</i> The name of the master table. <i>detailtable</i> The name of the detail table.
Example:	<pre>dbpEndRelationship "DB1" "Orders" "OrderDetails"</pre>
SQL Equivalent:	N/A

Grids

dbpShowGrid

Purpose:	Display a table in a grid format. Use an existing VisualNEO for Windows Rectangle object to serve as host for the grid. You can customize the appearance of the grid using dbpSetGridProperties , dbpSetGridBackground , dbpSetFieldProperties , dbpSetColumnTitles , dbpSetColumnWidths and dbpSetRowHeight .															
Category:	Grids															
Syntax:	dbpShowGrid "database id" "table" "rectangle" <i>database id</i> The name assigned to the database containing the table you want to display. <i>table</i> The name of the table to display. <i>rectangle</i> The name of an existing VisualNEO for Windows Rectangle object. The Rectangle will serve as a host for the table grid. Use VisualNEO for Windows's Tool Palette to create a Rectangle object on the page in your publication where you want the grid to appear. After executing dbpShowGrid, the grid will appear within the boundaries of the Rectangle object. The grid is only visible when the publication is running. Rectangle object before dbpShowGrid:  Rectangle object after dbpShowGrid: <table><thead><tr><th>FirstName</th><th>LastName</th><th>Street</th></tr></thead><tbody><tr><td>David</td><td>Thompson</td><td>128 East 14th St</td></tr><tr><td>Seth</td><td>Trenton</td><td>1588 Williams Court</td></tr><tr><td>Rose</td><td>Carney</td><td>6110 Westchester Ave</td></tr><tr><td>Harry</td><td>Eastman</td><td>12 North Freeson St</td></tr></tbody></table>	FirstName	LastName	Street	David	Thompson	128 East 14th St	Seth	Trenton	1588 Williams Court	Rose	Carney	6110 Westchester Ave	Harry	Eastman	12 North Freeson St
FirstName	LastName	Street														
David	Thompson	128 East 14th St														
Seth	Trenton	1588 Williams Court														
Rose	Carney	6110 Westchester Ave														
Harry	Eastman	12 North Freeson St														
Example:	dbpShowGrid "AddrBook" "Contacts" "Rectangle1"															
SQL Equivalent:	N/A															

dbpHideGrid

Purpose:	Remove a grid previously displayed with the dbpShowGrid action.
Category:	Grids
Syntax:	dbpHideGrid "database id" "table" <i>database id</i> The name assigned to the database containing the table. <i>table</i> The name of the table whose grid you want to hide.

Example:	dbpHideGrid "AddrBook" "Contacts"
SQL Equivalent:	N/A

dbpSetGridProperties

Purpose:	Customize the visual appearance and behavioral properties of a grid.																														
Category:	Grids																														
Syntax:	dbpSetGridProperties "database id" "table" "properties" <i>database id</i> The name assigned to the database containing the table. <i>table</i> The name of the table whose grid you want to customize. <i>properties</i> This is a compound parameter and can contain any combination of the following items: <table border="0"> <tr> <td>Color=<i>color</i></td><td>The color used for the background of the grid cells in odd numbered rows. See Defining Colors and dbpSetGridBackground.</td></tr> <tr> <td>Font=<i>font</i></td><td>The font name, size, style and character set of the font used for the grid cells. See Defining Fonts.</td></tr> <tr> <td>FontColor=<i>color</i></td><td>The color used for the grid cell text in odd numbered rows. See Defining Colors.</td></tr> <tr> <td>AlternateRowColor=<i>color</i></td><td>The color used for the background of the grid cells in even numbered rows. See Defining Colors.</td></tr> <tr> <td>AlertnateRowFontColor=<i>color</i></td><td>The color used for the grid cell text in even numbered rows. See Defining Colors.</td></tr> <tr> <td>RowHeight=<i>number</i></td><td>The height (in pixels) of the grid's rows.</td></tr> <tr> <td>TitleColor=<i>topcolor+bottomcolor</i></td><td>The color used for the background of the column headings. This can be either a single color to create a solid fill or two colors to create a gradient fill. To create a gradient, separate the top and bottom colors with a plus (+) sign. See Defining Colors.</td></tr> <tr> <td>TitleFont=<i>font</i></td><td>The font name, size, style and character set of the font used for the column headings. See Defining Fonts.</td></tr> <tr> <td>TitleFontColor=<i>color</i></td><td>The color used for the column heading text. See Defining Colors.</td></tr> <tr> <td>TitleRowHeight=<i>number</i></td><td>The height (in pixels) of the column heading row.</td></tr> <tr> <td>HighlightColor=<i>topcolor+bottomcolor</i></td><td>The color used for the background of the selected cell. This can be either a single color to create a solid fill or two colors to create a gradient fill. To create a gradient, separate the top and bottom colors with a plus (+) sign. See Defining Colors.</td></tr> <tr> <td>HighlightFontColor=<i>color</i></td><td>The color used for the selected cell's text. See Defining Colors.</td></tr> <tr> <td>EditColor=<i>color</i></td><td>The color used for the background of the cell being edited. See Defining Colors.</td></tr> <tr> <td>EditFontColor=<i>color</i></td><td>The color used for the text in the cell being edited. See Defining Colors.</td></tr> <tr> <td>ShowTitles=<i>yes/no</i></td><td>Yes = display the column heading row. No = do not display the column heading row.</td></tr> </table>	Color= <i>color</i>	The color used for the background of the grid cells in odd numbered rows. See Defining Colors and dbpSetGridBackground .	Font= <i>font</i>	The font name, size, style and character set of the font used for the grid cells. See Defining Fonts .	FontColor= <i>color</i>	The color used for the grid cell text in odd numbered rows. See Defining Colors .	AlternateRowColor= <i>color</i>	The color used for the background of the grid cells in even numbered rows. See Defining Colors .	AlertnateRowFontColor= <i>color</i>	The color used for the grid cell text in even numbered rows. See Defining Colors .	RowHeight= <i>number</i>	The height (in pixels) of the grid's rows.	TitleColor= <i>topcolor+bottomcolor</i>	The color used for the background of the column headings. This can be either a single color to create a solid fill or two colors to create a gradient fill. To create a gradient, separate the top and bottom colors with a plus (+) sign. See Defining Colors .	TitleFont= <i>font</i>	The font name, size, style and character set of the font used for the column headings. See Defining Fonts .	TitleFontColor= <i>color</i>	The color used for the column heading text. See Defining Colors .	TitleRowHeight= <i>number</i>	The height (in pixels) of the column heading row.	HighlightColor= <i>topcolor+bottomcolor</i>	The color used for the background of the selected cell. This can be either a single color to create a solid fill or two colors to create a gradient fill. To create a gradient, separate the top and bottom colors with a plus (+) sign. See Defining Colors .	HighlightFontColor= <i>color</i>	The color used for the selected cell's text. See Defining Colors .	EditColor= <i>color</i>	The color used for the background of the cell being edited. See Defining Colors .	EditFontColor= <i>color</i>	The color used for the text in the cell being edited. See Defining Colors .	ShowTitles= <i>yes/no</i>	Yes = display the column heading row. No = do not display the column heading row.
Color= <i>color</i>	The color used for the background of the grid cells in odd numbered rows. See Defining Colors and dbpSetGridBackground .																														
Font= <i>font</i>	The font name, size, style and character set of the font used for the grid cells. See Defining Fonts .																														
FontColor= <i>color</i>	The color used for the grid cell text in odd numbered rows. See Defining Colors .																														
AlternateRowColor= <i>color</i>	The color used for the background of the grid cells in even numbered rows. See Defining Colors .																														
AlertnateRowFontColor= <i>color</i>	The color used for the grid cell text in even numbered rows. See Defining Colors .																														
RowHeight= <i>number</i>	The height (in pixels) of the grid's rows.																														
TitleColor= <i>topcolor+bottomcolor</i>	The color used for the background of the column headings. This can be either a single color to create a solid fill or two colors to create a gradient fill. To create a gradient, separate the top and bottom colors with a plus (+) sign. See Defining Colors .																														
TitleFont= <i>font</i>	The font name, size, style and character set of the font used for the column headings. See Defining Fonts .																														
TitleFontColor= <i>color</i>	The color used for the column heading text. See Defining Colors .																														
TitleRowHeight= <i>number</i>	The height (in pixels) of the column heading row.																														
HighlightColor= <i>topcolor+bottomcolor</i>	The color used for the background of the selected cell. This can be either a single color to create a solid fill or two colors to create a gradient fill. To create a gradient, separate the top and bottom colors with a plus (+) sign. See Defining Colors .																														
HighlightFontColor= <i>color</i>	The color used for the selected cell's text. See Defining Colors .																														
EditColor= <i>color</i>	The color used for the background of the cell being edited. See Defining Colors .																														
EditFontColor= <i>color</i>	The color used for the text in the cell being edited. See Defining Colors .																														
ShowTitles= <i>yes/no</i>	Yes = display the column heading row. No = do not display the column heading row.																														

3DTitles= <i>yes/no</i>	Yes = add a 3D shadow effect to the column heading row and indicator column. No = no 3D effect.								
ShowGraphics= <i>yes/no</i>	Yes = display picture fields in the grid. No = don't display picture fields.								
ShowIndicator= <i>yes/no</i>	Yes = display an indicator along the left side of the grid identifying the active row. No = do not display an indicator.								
ShowColumnLines= <i>yes/no</i>	Yes = draw vertical lines between columns. No = do not draw lines between columns.								
ShowRowLines= <i>yes/no</i>	Yes = draw horizontal lines between rows. No = do not draw lines between rows.								
GridLineWidth= <i>number</i>	The thickness (in pixels) of the grid lines.								
GridLineColor= <i>color</i>	The color of the grid lines. See Defining Colors .								
AllowRowResize= <i>yes/no</i>	Yes = user may use the mouse to manually adjust the row height. No = user is prohibited from adjusting the row height.								
AllowColumnResize= <i>yes/no</i>	Yes = user may use the mouse to manually adjust the column widths. No = user is prohibited from adjusting the column widths.								
AllowColumnSort= <i>yes/no</i>	Yes = the user can sort the table by clicking on the column headings. No = user cannot sort the table.								
ConfirmDelete= <i>yes/no</i>	Yes = requires user confirmation when clicking the delete button in a grid's navigation bar. No = no confirmation required for deletes. The confirmation dialog can be customized using the dbpTranslateHints action.								
ReadOnly= <i>yes/no</i>	Yes = the table cannot be edited from the grid. No = grid allows editing.								
EditOnly= <i>yes/no</i>	Yes = existing records in the table can be edited, but no new records can be added from the grid. No = allow new records to be added from the grid.								
RowSelect= <i>yes/no</i>	Yes = the entire row will appear selected. No = only individual cells can be selected. When RowSelect=Yes, the grid does not permit editing.								
ShowScrollBars= <i>value</i>	This option determines whether the grid includes horizontal and vertical scroll bars. Value may be one of the following: <table> <tr> <td>Auto</td><td>Display both horizontal and vertical scroll bars (if needed).</td></tr> <tr> <td>Horizontal</td><td>Display only the horizontal scroll bar (if needed).</td></tr> <tr> <td>Vertical</td><td>Display only the vertical scroll bar (if needed).</td></tr> <tr> <td>None</td><td>Do not display scroll bars.</td></tr> </table>	Auto	Display both horizontal and vertical scroll bars (if needed).	Horizontal	Display only the horizontal scroll bar (if needed).	Vertical	Display only the vertical scroll bar (if needed).	None	Do not display scroll bars.
Auto	Display both horizontal and vertical scroll bars (if needed).								
Horizontal	Display only the horizontal scroll bar (if needed).								
Vertical	Display only the vertical scroll bar (if needed).								
None	Do not display scroll bars.								
ShowNavigationBar= <i>yes/no</i>	Yes = display a navigation bar at the bottom of the grid. No = do not display a navigation bar.								
GetDefaultValues= <i>yes/no</i>	Yes = when a new record is created, attempt to retrieve the default field values from the database server. No = do not retrieve the default values.								
Note: Normally, a database server will not apply default field values (defined with dbpCreateTable) until a record is posted. When using a grid to insert new records, this behavior causes fields to appear empty instead of displaying their default values. Only when the record is									

	<i>posted will the default values display.</i> <i>When GetDefaultValues = Yes, NeoDBpro will attempt to retrieve and apply the default values for each field at the time the record is inserted. Setting GetDefaultValues to "No" tells NeoDBpro to let the database server handle default field values normally.</i> <i>You can override the server's default field values with the DefaultValue property of the dbpSetFieldProperties action.</i> The name of a subroutine from your publication's Subroutine Action. The subroutine specified here will be executed whenever the user double clicks anywhere within the grid. You can use the dbpGetActiveField action in the subroutine to determine which field was clicked if needed. Subroutines are entered from the Actions page of VisualNEO for Windows's Book Properties screen. Separate multiple items in a compound variable with semicolons (;). Because of the large number of options, it is recommended that you use the wizard provided for the dbpSetGridProperties Action.
Example:	dbpSetGridProperties "AddrBook" "Contacts" "Color=White;AlternateRowColor=Silver" dbpShowGrid "AddrBook" "Contacts" "Rectangle1"
SQL Equivalent:	N/A

dbpSetGridBackground

Purpose:	Specify an image to serve as the grid's background. The background image will override the background color settings specified in dbpSetGridProperties . If the image is smaller than the grid, it will be tiled. To restore the grid's normal appearance, leave the file name parameter blank.
Category:	Grids
Syntax:	dbpSetGridBackground "database id" "table" "image file" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>image file</i> The name of the image file to use as the grid's background. Supported image formats are bmp, gif, tiff, png and jpeg. Leave this parameter blank to clear the background image.
Example:	dbpSetGridBackground "AddrBook" "Contacts" "C:\Images\PolishedSteel.png"
SQL Equivalent:	N/A

dbpSetFieldProperties

Purpose:	Define the formatting and edit properties of a field.
Category:	Grids

Syntax:	dbpSetFieldProperties "database id" "table" "field" "properties" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table containing the field you want to modify. <i>field</i> The name of the field to modify. <i>properties</i> This is a compound parameter and can contain any combination of the following items: <table> <tr> <td>Alignment=<i>value</i></td><td>Alignment method used to display the field's contents. Can be either Left, Right or Center.</td></tr> <tr> <td>TitleAlignment=<i>value</i></td><td>Alignment method used to display the field's column header. Can be either Left, Right or Center.</td></tr> <tr> <td>PickList=<i>list items</i></td><td>A list of possible values that can be selected by the user for input into the field. When editing in grid mode, a field that has been assigned a pick list will function like a combo box allowing the user to select list items from a drop down window.</td></tr> <tr> <td>DropDownRows=<i>number</i></td><td>The size (in lines) of the PickList drop down window.</td></tr> <tr> <td>DisplayFormat=<i>value</i></td><td>This option can be used to specify the display format used for date/time and numeric field types. The display format affects the appearance of the field values in both the grid and VisualNEO for Windows field variables. See Formatting Fields for more information.</td></tr> <tr> <td>EditMask=<i>mask</i></td><td>A validation mask used to restrict what can be entered into the field. Any invalid characters entered are rejected. Leave this item blank to allow any characters to be entered. See "Text Entry Field Tool > Validation Mask" in VisualNEO for Windows's Help file for information on constructing an edit mask.</td></tr> <tr> <td>DefaultValue=<i>value</i></td><td>A default value to be assigned to the field when a new record is created. This value must be compatible with the field's type.</td></tr> <tr> <td></td><td>Note: This value takes precedence over any default values for the field implemented on the database server (defined with dbpCreateTable) See also dbpSetGridProperties > GetDefaultValues.</td></tr> <tr> <td>ValidChars=<i>chars</i></td><td>A list of characters (letters, numbers or punctuation) that can be entered into the field. Leave this item blank to allow any characters to be entered.</td></tr> <tr> <td>ColumnWidth=<i>number</i></td><td>The width (in pixels) of the field's column in the grid.</td></tr> <tr> <td>ReadOnly=<i>yes/no</i></td><td>Yes = the field cannot be edited from the grid. No = grid allows editing.</td></tr> <tr> <td>Visible=<i>yes/no</i></td><td>Yes = the field is displayed in the grid. No = the field does not appear in the grid.</td></tr> </table> Separate multiple items in a compound variable with semicolons (;).	Alignment= <i>value</i>	Alignment method used to display the field's contents. Can be either Left, Right or Center.	TitleAlignment= <i>value</i>	Alignment method used to display the field's column header. Can be either Left, Right or Center.	PickList= <i>list items</i>	A list of possible values that can be selected by the user for input into the field. When editing in grid mode, a field that has been assigned a pick list will function like a combo box allowing the user to select list items from a drop down window.	DropDownRows= <i>number</i>	The size (in lines) of the PickList drop down window.	DisplayFormat= <i>value</i>	This option can be used to specify the display format used for date/time and numeric field types. The display format affects the appearance of the field values in both the grid and VisualNEO for Windows field variables. See Formatting Fields for more information.	EditMask= <i>mask</i>	A validation mask used to restrict what can be entered into the field. Any invalid characters entered are rejected. Leave this item blank to allow any characters to be entered. See "Text Entry Field Tool > Validation Mask" in VisualNEO for Windows's Help file for information on constructing an edit mask.	DefaultValue= <i>value</i>	A default value to be assigned to the field when a new record is created. This value must be compatible with the field's type.		Note: This value takes precedence over any default values for the field implemented on the database server (defined with dbpCreateTable) See also dbpSetGridProperties > GetDefaultValues .	ValidChars= <i>chars</i>	A list of characters (letters, numbers or punctuation) that can be entered into the field. Leave this item blank to allow any characters to be entered.	ColumnWidth= <i>number</i>	The width (in pixels) of the field's column in the grid.	ReadOnly= <i>yes/no</i>	Yes = the field cannot be edited from the grid. No = grid allows editing.	Visible= <i>yes/no</i>	Yes = the field is displayed in the grid. No = the field does not appear in the grid.
Alignment= <i>value</i>	Alignment method used to display the field's contents. Can be either Left, Right or Center.																								
TitleAlignment= <i>value</i>	Alignment method used to display the field's column header. Can be either Left, Right or Center.																								
PickList= <i>list items</i>	A list of possible values that can be selected by the user for input into the field. When editing in grid mode, a field that has been assigned a pick list will function like a combo box allowing the user to select list items from a drop down window.																								
DropDownRows= <i>number</i>	The size (in lines) of the PickList drop down window.																								
DisplayFormat= <i>value</i>	This option can be used to specify the display format used for date/time and numeric field types. The display format affects the appearance of the field values in both the grid and VisualNEO for Windows field variables. See Formatting Fields for more information.																								
EditMask= <i>mask</i>	A validation mask used to restrict what can be entered into the field. Any invalid characters entered are rejected. Leave this item blank to allow any characters to be entered. See "Text Entry Field Tool > Validation Mask" in VisualNEO for Windows's Help file for information on constructing an edit mask.																								
DefaultValue= <i>value</i>	A default value to be assigned to the field when a new record is created. This value must be compatible with the field's type.																								
	Note: This value takes precedence over any default values for the field implemented on the database server (defined with dbpCreateTable) See also dbpSetGridProperties > GetDefaultValues .																								
ValidChars= <i>chars</i>	A list of characters (letters, numbers or punctuation) that can be entered into the field. Leave this item blank to allow any characters to be entered.																								
ColumnWidth= <i>number</i>	The width (in pixels) of the field's column in the grid.																								
ReadOnly= <i>yes/no</i>	Yes = the field cannot be edited from the grid. No = grid allows editing.																								
Visible= <i>yes/no</i>	Yes = the field is displayed in the grid. No = the field does not appear in the grid.																								
Example:	<pre>dbpSetFieldProperties "MySQL" "Table1" "[ListBox6]" "Alignment=Left;TitleAlignment=Left; PickList=Apple,Orange,Cherry,Grape,Pear,Strawberry; DropDownRows=7;EditMask=;ValidChars=;ColumnWidth=30;ReadOnly=No;Visible=Yes"</pre>																								
SQL Equivalent:	N/A																								

dbpGetActiveField

Purpose:	Returns the name of the field in the grid that is highlighted or active.
Category:	Grids
Syntax:	dbpGetActiveField "database id" "table" "variable" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>variable</i> The name of the variable where the name of the active field will be stored.
Example:	<pre>dbpGetActiveField "AddrBook" "Contacts" "[SelField]" If "[SelField]" ">" "" dbpSort "AddrBook" "Contacts" "[SelField] ASC" EndIf</pre>
SQL Equivalent:	N/A

dbpSetColumnTitles

Purpose:	Customize the titles that appear at the top of each column in a grid. A grid contains one vertical column for each field in the table. By default, the column titles are the same as the field names, which can sometimes be a little cryptic. You can use this action to change only the column titles without affecting the names of the fields.
Category:	Grids
Syntax:	dbpSetColumnTitles "database id" "table" "titles" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>titles</i> A list containing the original field name, an equal sign and the title to be assigned to each column. Separated multiple items with a semicolon character (;). For example, the column titles for a table containing six fields (FIRSTNAME, LASTNAME, STREET, CITY, STATE, ZIP) might look like this: "FIRSTNAME=First Name;LASTNAME=Last Name;STREET=Street;CITY=City;STATE=State;ZIP=Postal Code" It is not necessary to include every field in the list. You can shorten the action by including only the field's that you want to change. Field's not in the list will retain their existing column titles. For example: "FIRSTNAME=First Name;LASTNAME=Last Name;ZIP=Postal Code"
Example:	<pre>dbpSetColumnTitles "AddrBook" "Contacts" "FirstName=First Name;LastName=Last Name;State=State/Prov;Zip=Postal Code"</pre>
SQL Equivalent:	N/A

dbpSetColumnOrder

Purpose:	Specify the order in which columns/fields appear in the grid.
Category:	Grids
Syntax:	<code>dbpSetColumnOrder "database id" "table" "field list"</code> <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>field list</i> A list of the field names in the order you want them to appear in the grid. Each field name must be separated by a semicolon character (;). You can hide a field by omitting its name from the list.
Example:	<code>dbpSetColumnOrder "AddrBook" "Contacts"</code> <code>"LastName;FirstName;Company;City;Street;State;Zip;Country;Telephone;Email;WebSite;Comments"</code>
SQL Equivalent:	N/A

dbpSetColumnWidths

Purpose:	Set the width of each column/field in the grid. If you have enabled the AllowColumnResize option in dbpSetGridProperties , you can combine this action with dbpGetColumnWidths to save and restore changes users make to grid columns.
Category:	Grids
Syntax:	<code>dbpSetColumnWidths "database id" "table" "field list"</code> <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>field list</i> A list containing the original field name, an equal sign and the width (in pixels) to be assigned to each column. Separated multiple items with a semicolon character (;). This is the same format returned by dbpGetColumnWidths . For example, the column with for a table containing six fields (FIRSTNAME, LASTNAME, STREET, CITY, STATE, ZIP) might look like this: <code>"FIRSTNAME=50;LASTNAME=50;STREET=120;CITY=25;STATE=30;ZIP=35"</code> It is not necessary to include every field in the list. You can shorten the action by including only the field's that you want to change. Field's not in the list will retain their existing column widths. For example: <code>"FIRSTNAME=50;LASTNAME=50;STATE=30"</code>
Example:	<code>dbpSetColumnWidths "AddrBook" "Contacts"</code> <code>"FirstName=90;LastName=90;Company=90;Street=150;City=90;State=90;Zip=90;Country=90;Telephone=90;Email=90;WebSite=90;Comments=350"</code>
SQL Equivalent:	N/A

dbpGetColumnWidths

Purpose:	Get the width of each column/field in the grid.
Category:	Grids
Syntax:	<code>dbpGetColumnWidths "database id" "table" "variable"</code> <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>variable</i> The name of the variable where the column widths will be stored. The variable format will be identical to that used by the dbpSetColumnWidths action.
Example:	<code>dbpGetColumnWidths "AddrBook" "Contacts" "[ColWidths]"</code>
SQL Equivalent:	N/A

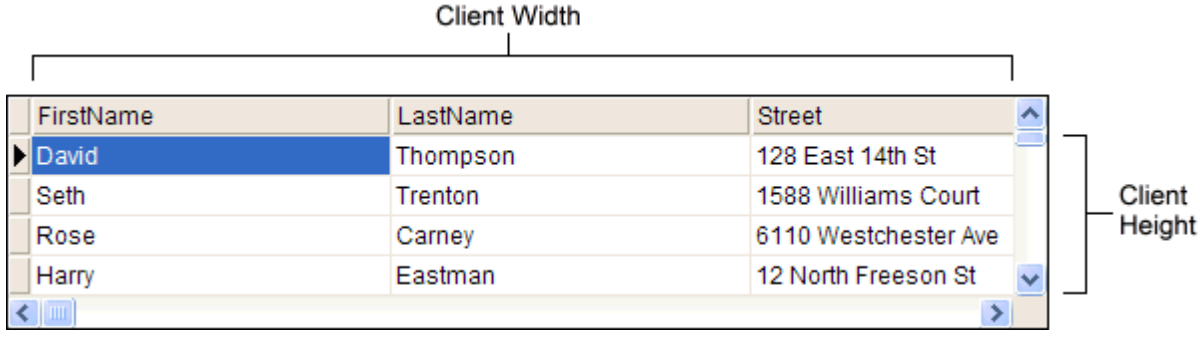
dbpSetRowHeight

Purpose:	Set the height of all rows in the grid. Use dbpGetRowHeight to obtain the default row height.
Category:	Grids
Syntax:	<code>dbpSetRowHeight "database id" "table" "height"</code> <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>height</i> The height (in pixels) for all rows in the grid.
Example:	<code>dbpSetRowHeight "AddrBook" "Contacts" "26"</code>
SQL Equivalent:	N/A

dbpGetRowHeight

Purpose:	Obtain the height of a single grid row. All rows in the grid are the same height, so only one number is returned. If you have enabled the AllowRowResize option in dbpSetGridProperties , this action can be combined with dbpSetRowHeight to save and restore changes to the row height made by users during the execution of your publication.
Category:	Grids
Syntax:	<code>dbpGetRowHeight "database id" "table" "variable"</code> <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>variable</i> The name of the variable where the row height will be stored.
Example:	<code>dbpGetRowHeight "AddrBook" "Contacts" "[RowHeight]"</code>
SQL Equivalent:	N/A

dbpGetGridClientInfo

Purpose:	Obtain the width and height of the grid's client/display area. This information can be useful for calculating column widths and row heights. The client area includes the display portion of the grid excluding the indicator column and the scroll bars.  Note: Settings applied to the grid with dbpSetGridProperties will be considered when calculating the client width and height.
Category:	Grids
Syntax:	<pre>dbpGetGridClientInfo "database id" "table" "width variable" "height variable"</pre> <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>width variable</i> The name of the variable where the grid's client width will be stored. <i>height variable</i> The name of the variable where the grid's client height will be stored.
Example:	<code>dbpGetGridClientInfo "AddrBook" "Contacts" "[cw]" "[ch]"</code>
SQL Equivalent:	N/A

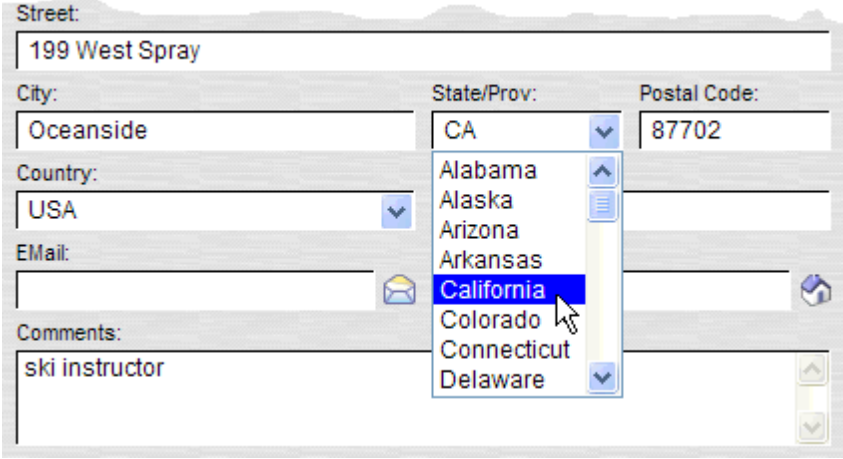
dbpDefineValueList

Purpose:	Identify a field from one table to be used as an input source for a field in another table. When editing in grid mode, a field that has been assigned a value list will function like a combo box allowing the user to select list items from a drop down window. This is similar to the <code>dbpSetGridProperties</code>' <code>PickList</code> option except that the contents of the list are derived from the contents of another table.
Category:	Grids
Syntax:	<pre>dbpDefineValueList "database id" "main table" "edit field" "list table" "data field" "display field"</pre> <i>database id</i> The name assigned to the database. <i>main table</i> The name of the main table containing the field to be edited. <i>edit field</i> The name of the field in the main table to receive input from the value list. <i>list table</i>

	The name of the table containing the field to be used as the source of the value list. <i>data field</i> The field in the source table containing the actual data that will be input into the edit field. <i>display field</i> The field or fields in the source table to be displayed in the value list combo box. This can be the same as the data field or a different field. Separate multiple display fields with semicolons (;).																																												
Example:	The following example defines the contents of the States table as a value list for the State field of the Contacts table: <pre>dbpDefineValueList "AddrBook" "Contacts" "State" "States" "Abbr" "Name"</pre> When editing the Contacts table in grid mode, clicking on the State field will automatically display a list of states drawn from the contents of the States table: <table><tr><th>City</th><th>State/Prov</th><th>Postal Code</th><th>Country</th></tr><tr><td>Oceanside</td><td>CA</td><td>87702</td><td>USA</td></tr><tr><td>Salem</td><td>MA</td><td>68880</td><td>USA</td></tr><tr><td>Breckenridge</td><td>Colorado</td><td>87758</td><td>USA</td></tr><tr><td>Williamsburg</td><td>Alaska</td><td>37798</td><td>USA</td></tr><tr><td>Phoenix</td><td>Arizona</td><td>89974</td><td>USA</td></tr><tr><td>Clarkston</td><td>Arkansas</td><td>88520</td><td>USA</td></tr><tr><td>New Bedford</td><td>California</td><td>17792</td><td>USA</td></tr><tr><td>Home Town</td><td>Colorado</td><td>90032</td><td>USA</td></tr><tr><td></td><td>Connecticut</td><td></td><td></td></tr><tr><td></td><td>Delaware</td><td></td><td></td></tr></table>	City	State/Prov	Postal Code	Country	Oceanside	CA	87702	USA	Salem	MA	68880	USA	Breckenridge	Colorado	87758	USA	Williamsburg	Alaska	37798	USA	Phoenix	Arizona	89974	USA	Clarkston	Arkansas	88520	USA	New Bedford	California	17792	USA	Home Town	Colorado	90032	USA		Connecticut				Delaware		
City	State/Prov	Postal Code	Country																																										
Oceanside	CA	87702	USA																																										
Salem	MA	68880	USA																																										
Breckenridge	Colorado	87758	USA																																										
Williamsburg	Alaska	37798	USA																																										
Phoenix	Arizona	89974	USA																																										
Clarkston	Arkansas	88520	USA																																										
New Bedford	California	17792	USA																																										
Home Town	Colorado	90032	USA																																										
	Connecticut																																												
	Delaware																																												
SQL Equivalent:	N/A																																												

dbpPopupValueList

Purpose:	Display a popup window containing a list of lookup items derived from a field in a table. This action is similar to dbpDefineValueList above, except that it does not require a grid. The popup value list can be used anywhere allowing the user to select list items from a drop down window. The selected item is stored in a variable.
Category:	Grids
Syntax:	<pre>dbpPopupValueList "database id" "list table" "data field" "display field" "properties" "variable"</pre> <i>database id</i> The name assigned to the database. <i>list table</i> The name of the table containing the field to be used as the source of the value list. <i>data field</i> The field in the source table containing the actual data that will be stored in the variable. <i>display field</i> The field or fields in the source table to be displayed in the popup window. This can be the same as the data field or a different field. Separate multiple display fields with semicolons (;). <i>properties</i> This is a compound parameter and can contain any combination of the following items:

	<i>Left=number</i> The coordinates of the popup window's left corner (in pixels) relative to the publication window. <i>Top=number</i> The coordinates of the popup window's top corner (in pixels) relative to the publication window. <i>Width=number</i> The width (in pixels) of the popup window. <i>RowCount=number</i> The number of lines to display in the list. Separate multiple items in a compound variable with semicolons (;).
Example:	The following example displays a list of state names (display field) from the States table: <pre>dbpPopupValueList "AddrBook" "States" "Abbr" "Name" "Left=216;Top=160;Width=96;RowCount=15" "[AddrBook.Contacts.State]"</pre> The returned value will be the corresponding value from the Abbr field (data field). For example, if the user selects "California" from the popup list, the value stored in the variable will be "CA". 
SQL Equivalent:	N/A

Created with the Standard Edition of HelpNDoc: [Easily create Help documents](#)

Navigation

dbpNext

Purpose:	Display the next record in the table. When a query is active, only records matching the search criteria will be displayed.
Category:	Navigation
Syntax:	<pre>dbpNext "database id" "table"</pre> <i>database id</i> The name assigned to the database. <i>table</i> The name of the table.
Example:	dbpNext "AddrBook" "Contacts"
SQL Equivalent:	N/A

dbpPrev

Purpose:	Display the previous record in the table. When a query is active, only records matching the search criteria will be displayed.
Category:	Navigation
Syntax:	dbpPrev "database id" "table" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table.
Example:	dbpPrev "AddrBook" "Contacts"
SQL Equivalent:	N/A

dbpFirst

Purpose:	Display the first record in the table. When a query is active, only records matching the search criteria will be displayed.
Category:	Navigation
Syntax:	dbpFirst "database id" "table" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table.
Example:	dbpFirst "AddrBook" "Contacts"
SQL Equivalent:	N/A

dbpLast

Purpose:	Display the last record in the table. When a query is active, only records matching the search criteria will be displayed.
Category:	Navigation
Syntax:	dbpLast "database id" "table" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table.
Example:	dbpLast "AddrBook" "Contacts"
SQL Equivalent:	N/A

dbpGotoRecord

Purpose:	Display a specific record based on its sequential position in the table. Note: A record's relative position in the table may change depending on the active query
----------	---

	<i>and sort settings.</i>
Category:	Navigation
Syntax:	dbpGotoRecot "database id" "table" "record num" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>record num</i> The index number of the record you want to display. This is the record's sequential position in the database based on the current query and sort settings (first record = 1).
Example:	dbpGotoRecord "AddrBook" "Contacts" "10"
SQL Equivalent:	N/A

Created with the Standard Edition of HelpNDoc: [Easy CHM and documentation editor](#)

Search/Query

dbpQuery

Purpose:	Limit the display of records to those that match specific search criteria. This action is similar to dbpFind, but dbpQuery allows you to search multiple fields at the same time with more complex search criteria. After calling dbpQuery, only records matching the search criteria will be visible. You can examine the special variable [ID.Table.\$RecCount] to determine the number of records returned by the query. Navigation actions, such as dbpFirst, dbpPrev, dbpNext and dbpLast, will only display records that fall within the query's search parameters. To cancel the query and display all records, use the dbpShowAll action.
Category:	Search/Query
Syntax:	dbpQuery "database id" "table" "filter" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table to query. <i>filter</i> The filter defines the parameters of your query. The filter generally consists of three elements - a field name, an operator and search string. The field name is the name of the field you want to search. Search string is the text or data you want to find. The operator can be one of the following: = The search string matches the field contents exactly. For example: City = "Pittsburgh" LIKE The search string is similar to the field contents. The LIKE operator allows you to include wildcards to find partial matches. For example, the following will find all customers having a first name that starts with the letter J: FirstName LIKE "J%" The % character is a wildcard. The wildcard can be placed anywhere in the string and you can use as many wildcards as you like.

	Here are some examples: LastName LIKE "%duff%" will find "Duffy", "Macduff", "Duffel", etc. Phone LIKE "416%" will find telephone numbers beginning with 416. Another wildcard is the underscore character (_) which can be used to represent any single character. For example: LastName LIKE "sm_th" will find "Smith", "Smyth", "Smath", "Smeth", etc.
<	The contents of the field is less than the search string. This is most often used with numeric fields. For example: Salary < 50000
>	The contents of the field is greater than the search string. This is most often used with numeric fields. For example: Salary > 50000
<=	The contents of the field is less than or equal to the search string. This is most often used with numeric fields. For example: Age <= 65
>=	The contents of the field is greater than or equal to the search string. This is most often used with numeric fields. For example: Age >= 18
<>	The search string does not match the contents of the field. This is most often used with numeric fields. For example: Height <> 6.1
BETWEEN	The contents of the field falls between two values. This is most often used with numeric fields. For example: Age BETWEEN 18 AND 34 Note: For string type fields, data must surrounded by double quotes. For example: Age BETWEEN "18" AND "34"
IN	The contents of the field matches a specific set of values. For example: Age IN (18,19,28,29,38,39) Note: For string type fields, data must surrounded by double quotes. For example: Age IN ("18","19","28","29","38","39") For example, the following filter will find all records where the Salary field equals \$50,000: "Salary = 50000"

	If the field being searched is a String, Char or Memo type then the search string must be surrounded by quotes which, in VisualNEO for Windows, are specified using the special code: [#34]. For example: <pre>"City = [#34]Pittsburgh[#34]"</pre> You can also construct more complex queries by combining more than one filter. For example, to find all records where the City field equals either Pittsburgh OR St Louis: <pre>"City = [#34]Pittsburgh[#34] OR City=[#34]St. Louis[#34]"</pre> To find records where the City field equals Pittsburgh and the LastName field equals Jones <pre>"City = [#34]Pittsburgh[#34] AND LastName = [#34]Jones[#34]"</pre> The < and > operators are Primarily useful when searching fields that contain numbers. For example, to Search our inventory database for items costing less than \$10, we might use a filter like this: <pre>"Parts < 10.00"</pre> At times you may want to find records that don't match certain criteria. For this you can add the special operator "NOT". For example: <pre>"NOT Country = [#34]USA[#34]"</pre> There are certain situations where you might want to find fields that are empty. To do this you can use the special operator "IS" and the keyword "NULL". For example: <pre>"Country IS NULL"</pre> Or to find non empty fields: <pre>"Country IS NOT NULL"</pre> Note: Advanced users familiar with SQL can perform more advanced types of queries using the <code>dbpExecSQL</code> action.
Example:	<pre>dbpQuery "AddrBook" "Contacts" "State = [#34]CA[#34]"</pre> You can use the special variable [ID.Table.\$RecCount] to determine if the query was successful. For example: <pre>dbpQuery "AddrBook" "Contacts" "State = [#34]CA[#34]" If "[AddrBook.Contacts.\$RecCount]" "=" "0" MsgBox "Error" "No records matching the search criteria were found." dbpShowAll "AddrBook" "Contacts" EndIf</pre>
SQL Equivalent:	<pre>SELECT * FROM table WHERE filter</pre>

dbpShowAll

Purpose:	Discontinue current search query and display all records. See dbpQuery .
Category:	Search/Query

Syntax:	dbpShowAll "database id" "table" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table.
Example:	dbpShowAll "AddrBook" "Contacts"
SQL Equivalent:	SELECT * FROM <i>table</i>

dbpFind

Purpose:	Search one or more fields for a specific string or value and display the first matching record. Use dbpFindNext to display additional matching records. See dbpQuery for more complex searches.
Category:	Search/Query
Syntax:	dbpFind "database id" "table" "fields" "search string" "options" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>fields</i> A single field or list of fields to search. Multiple fields will be searched in the order that they appear. Separate multiple fields with semicolons (;). <i>search string</i> The text or value you want to find. <i>options</i> This is a compound parameter and can contain any combination of the following items: ExactMatch=Yes/No Yes = search string and the contents of the field must match exactly. No = a partial match is acceptable. CaseSensitive=Yes/No Yes = upper and lower case letters in the search string must be identical to the field contents. No = ignore case. Separate multiple items in a compound variable with semicolons (;). This parameter may be left blank if you do not require either of these options.
Example:	dbpFind "AddrBook" "Contacts" "FirstName;LastName;Company" "[SearchStr]" "ExactMatch=No;CaseSensitive=No"
SQL Equivalent:	N/A

dbpFindNext

Purpose:	Search for the next record that matched the previous dbpFind criteria. dbpFind must be called at least once before calling dbpFindNext.
Category:	Search/Query
Syntax:	dbpFindNext "database id" "table" <i>database id</i> The name assigned to the database.

	<i>table</i> The name of the table.
Example:	dbpFindNext "AddrBook" "Contacts"
SQL Equivalent:	N/A

Created with the Standard Edition of HelpNDoc: [Easy EBook and documentation generator](#)

Advanced SQL

dbpExecSQL

Purpose:	Execute advanced Structured Query Language (SQL) commands. You should be able to execute many types of SQL commands using this action. However, keep in mind that there are often differences in features and syntax between different databases. It's a good idea to refer to your database documentation to insure that the SQL commands you are planning to use are fully supported.
Category:	AdvancedSQL
Syntax:	dbpExecSQL "database id" "SQL" "results table" <i>database id</i> The name assigned to the database where the SQL commands will be executed. <i>SQL</i> The SQL commands to execute. As required by SQL conventions, multiple commands must be separated with semicolons (;). <i>results table (optional)</i> This parameter can be used to enter the name of the table where you want the results of the query to be displayed. This may be the name of an existing database table or it can be a temporary table. If the table specified does not exist in the database, it is assumed to be a temporary table. Leave this parameter blank to auto detect the table name or if the query does not return any results. A temporary table exists only while your publication is running and is not physically part of the database. However, most temporary tables derive their contents from real data, so any edits you make may affect other tables. A temporary table created with dbpExecSQL can be opened with the dbpOpenTable and displayed with dbpShowGrid actions. Actions that require a physical table, such as dbpAddField or dbpQuery , cannot be used with a temporary table.
Example:	dbpExecSQL "DB1" "RENAME TABLE Sales TO Orders" "" The following example performs a simple query and sends the results to a temporary table, which is then opened and displayed in a grid: <pre>dbpExecSQL "AddrBook" "SELECT * FROM Contacts WHERE State = 'NY'" "Temp1" dbpOpenTable "AddrBook" "Temp1" "" dbpShowGrid "AddrBook" "Temp1" "Rectangle1"</pre>
SQL Equivalent:	N/A

Created with the Standard Edition of HelpNDoc: [Create iPhone web-based documentation](#)

Sort/Index

dbpSort

Purpose:	Sort a table based on a specific field or group of fields. By default table records are usually displayed in the order that they were created. This action allows you to temporarily rearrange the records and display them in a different order. Fields can be sorted in either ascending or descending order. If an index for a sort field exists, it will be used automatically to speed the sort process. Navigation actions such as dbpFirst, dbpPrev, dbpNext and dbpLast actions will display records according the sort order. To restore the original record order and remove the sort, use dbpUnSort.
Category:	Sort/Index
Syntax:	dbpSort "database id" "table" "sort info" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table to sort. <i>sort info</i> The name of the field or group of fields to use for the sort. Each field name must be followed by an equal sign and a direction keyword (ASC for ascending order or DESC for descending order). If multiple fields are used they should be separated by semicolons (;). For example, the following will sort by State in ascending order (A-Z), then sort by City in descending order (Z-A): "State=ASC;City=DESC"
Example:	dbpSort "AddrBook" "Contacts" "LastName=ASC"
SQL Equivalent:	SELECT * FROM <i>table</i> ORDER BY <i>sort info</i>

dbpUnSort

Purpose:	Remove the current sort criteria specified with dbpSort and display the table records in their original ordinal order.
Category:	Sort/Index
Syntax:	dbpUnSort "database id" "table" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table to unsort.
Example:	dbpUnSort "AddrBook" "Contacts"
SQL Equivalent:	Usually a sort can be removed by simply performing another query. For example: SELECT * FROM <i>table</i>

dbpGetIndexNames

Purpose:	Obtain a list of indexes associated with a table and store the results in a variable. Index names will separated by the specified delimiter.
----------	--

Category:	Sort/Index
Syntax:	<code>dbpGetIndexNames "database id" "table" "delimiter" "variable"</code> <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>delimiter</i> The character or characters used to separate the index names. <i>variable</i> The name of the variable where the index names will be stored. Each index name will be separated by the specified delimiter.
Example:	<code>dbpGetIndexNames "AddrBook" "Contacts" ";" "[Indexes]"</code>
SQL Equivalent:	<code>SHOW INDEX FROM <i>table</i></code>

dbpCreateIndex

Purpose:	Create an index to speed up sort and query operations.				
Category:	Sort/Index				
Syntax:	<code>dbpCreateIndex "database id" "table" "fields" "index name" "options"</code> <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>fields</i> A single field or list of fields to be used for the index. Separate multiple fields with semicolons (;). <i>index name</i> The name to be used for this index. <i>options</i> This is a compound parameter and can contain any combination of the following items: <table border="0"> <tr> <td>PrimaryIndex=Yes/No</td><td>Yes = indexed field will become the primary key field for the table. A primary key field cannot be empty (NULL). Each table can have only one primary Index.</td></tr> <tr> <td>Unique=Yes/No</td><td>Yes = create a unique index. No = do not create a unique index. A unique index requires all values in the index must be different. An error occurs if you try to add a new record with a data that matches an existing record.</td></tr> </table> Separate multiple items in a compound variable with semicolons (;). This parameter may be left blank if you do not require either of these options.	PrimaryIndex=Yes/No	Yes = indexed field will become the primary key field for the table. A primary key field cannot be empty (NULL). Each table can have only one primary Index.	Unique=Yes/No	Yes = create a unique index. No = do not create a unique index. A unique index requires all values in the index must be different. An error occurs if you try to add a new record with a data that matches an existing record.
PrimaryIndex=Yes/No	Yes = indexed field will become the primary key field for the table. A primary key field cannot be empty (NULL). Each table can have only one primary Index.				
Unique=Yes/No	Yes = create a unique index. No = do not create a unique index. A unique index requires all values in the index must be different. An error occurs if you try to add a new record with a data that matches an existing record.				
Example:	<code>dbpCreateIndex "AddrBook" "Contacts" "LastName;FirstName" "NameIdx" "PrimaryIndex=No;Unique=No"</code>				
SQL Equivalent:	Normally, indexes are created at the time the table itself is created, however the following command can be used to add indexes to existing tables: <code>CREATE INDEX <i>index name</i> ON <i>table</i> (<i>fields</i>)</code>				

dbpDropIndex

Purpose:	Delete an existing table index.
Category:	Sort/Index
Syntax:	dbpDropIndex "database id" "table" "index name" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table containing the index. <i>index name</i> The name of the index to be deleted.
Example:	dbpDropIndex "AddrBook" "Contacts" "NameIdx"
SQL Equivalent:	DROP INDEX <i>index name</i>

Created with the Standard Edition of HelpNDoc: [Easily create EBooks](#)

Add/Modify**dbpAddRecord**

Purpose:	Add a new empty record to a table.
Category:	Add/Modify
Syntax:	dbpAddRecord "database id" "table" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table where the record will be added.
Example:	dbpAddRecord "AddrBook" "Contacts"
SQL Equivalent:	INSERT INTO <i>table</i> (fields) VALUES(data)

dbpDeleteRecord

Purpose:	Delete the current record.
Category:	Add/Modify
Syntax:	dbpDeleteRecord "database id" "table" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table containing the record to delete.
Example:	dbpDeleteRecord "AddrBook" "Contacts"
SQL Equivalent:	DELETE FROM <i>Table</i> WHERE <i>filter</i> *

*Deleting records with SQL requires that the target record(s) be identified using a query filter. To delete a single record, you must construct the filter so that the only matching record is the one you want to delete.

dbpDeleteAll

Purpose:	Delete all records matching current search query (see dbpQuery). If no query is active, the entire table will be deleted. Use this action with caution - there is no way to restore deleted records!
Category:	Add/Modify
Syntax:	dbpDeleteAll "database id" "table" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table containing the records to delete.
Example:	<pre> dbpQuery "DB1" "Vendors" "AccountStatus = [#34]Paid[#34]" If "[DB1.Vendors.\$RecCount]" "=" "0" MsgBox "Sorry" "No records found." Else MsgBox "Delete" "Delete all found records?" "Yes?No" "[Result]" If "[Result]" "=" "1" dbpDeleteAll "DB1" "Vendors" EndIf EndIf dbpShowAll "DB1" "Vendors" </pre>
SQL Equivalent:	DELETE FROM <i>table</i>

dbpEmptyTable

Purpose:	Remove all records from a table. This does the same thing as dbpDeleteAll except on some databases this action will execute faster. Use this action with caution - there is no way to restore deleted records!
Category:	Add/Modify
Syntax:	dbpEmptyTable "database id" "table" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table to be emptied.
Example:	dbpEmptyTable "AddrBook" "Contacts"
SQL Equivalent:	TRUNCATE <i>table</i>

dbpStrReplace

Purpose:	Replace all occurrences of a string in a single field. If a query is active, only records matching the search criteria will be copied. You can use this action to perform a search and replace across an entire table.
Category:	Add/Modify
Syntax:	dbpStrReplace "database id" "table" "field" "sourcestr" "replacestr" <i>database id</i> The name assigned to the database.

	<i>table</i> The name of the table. <i>field</i> The name of the field to search. <i>sourcestr</i> The text to be replaced. <i>replacestr</i> The replacement text.
Example:	dbpStrReplace "AddrBook" "Contacts" "Telephone" "(503)" "(541)"
SQL Equivalent:	The following replaces all occurrences of "Athens" with "Paris" in the Contact table's City field: <pre>UPDATE Contacts SET City="Paris" where City="Athens"</pre>

dbpRefresh

Purpose:	Reload the latest table data from the database. Use with multi-user databases to retrieve updates made by other users.
Category:	Add/Modify
Syntax:	dbpRefresh "database id" "table" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table to be refreshed.
Example:	dbpRefresh "AddrBook" "Contacts"
SQL Equivalent:	N/A

Created with the Standard Edition of HelpNDoc: [Write EPub books for the iPad](#)

Advanced Add/Modify

dbpSetAutoEdit

Purpose:	Changes can be discarded using dbpCancelEdits . AutoEdit is set to on automatically when opening a table.
Category:	Advanced Add/Modify
Syntax:	dbpSetAutoEdit "database id" "table" "status" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>status</i> The status of the AutoEdit feature. Use Yes to turn AutoEdit on or No to turn it off.
Example:	dbpSetAutoEdit "AddrBook" "Contacts" "Yes"
SQL Equivalent:	N/A

dbpSaveEdits

Purpose:	Manually save changes to the current record. This Action is intended to be used in conjunction with dbpCancelEdits and dbpSetAutoEdit to manually control when table records are updated. However, you can also use this command whenever you want to manually save changes made to the current table record regardless of the status of dbpSetAutoEdit.
Category:	Advanced Add/Modify
Syntax:	dbpSaveEdits "database id" "table" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table.
Example:	dbpSaveEdits "AddrBook" "Contacts"
SQL Equivalent:	N/A

dbpCancelEdits

Purpose:	Abandon changes made to the current record. Any edits made to the current record are discarded and the record's previous field contents are restored. This Action is intended to be used in conjunction with dbpSaveEdits to manually control when table records are updated.
Category:	Advanced Add/Modify
Syntax:	dbpCancelEdits "database id" "table" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table.
Example:	dbpCancelEdits "AddrBook" "Contacts"
SQL Equivalent:	N/A

Created with the Standard Edition of HelpNDoc: [Full-featured EPub generator](#)

Simple Reporting**dbpListPrint**

Purpose:	Print all records matching current search query (see dbpQuery). If no query is active, the entire table will be printed. The report is printed in a tabular format optionally using the color and font settings applied to the current table grid. Custom headers and footers for the report may be created using dbpListAddHeader and dbpListAddFooter . The report may also be sorted by calling dbpSort prior to printing.
Category:	Simple Reporting
Syntax:	dbpListPrint "database id" "table" "properties" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table to be printed.

	<i>properties</i> This is a compound parameter and can contain any combination of the following items: PrintDialog=Yes/No Yes = display the standard Windows print dialog prior to printing. No = do not display the print dialog - report will be printed immediately. Orientation=Portrait/Landscape Portrait = print the report in portrait (vertical) mode. Landscape = print the report in landscape (horizontal) mode. GridLines=Yes/No Yes = draw lines between columns and rows. No = do not draw lines between columns and rows. ColumnTitles=Yes/No Yes = display the column heading row at the top of each page. No = do not display the column heading row. RecordNumbers=Yes/No Yes = display record numbers along the left side of the report. No = do not display record numbers. AutoWidth=Yes/No Yes = automatically adjust the width of each column to fit the page. No = use the column widths assigned to the table grid by dbpSetColumnWidths. UseGridAttributes=Yes/No Yes = use the colors, fonts and other properties assigned to the table grid with the dbpSetGridProperties action. No = do not use the grid properties. LeftMargin=value RightMargin=value TopMargin=value BottomMargin=value Page margins to be used for the report. Margins can be specified as inches or centimeters which is indicated by adding "in" or "cm" after the value. For example: "LeftMargin=1.5in" "LeftMargin=3.25cm" Separate multiple items in a compound variable with semicolons (;). Because of the large number of options, it is recommended that you use the wizard provided for the dbpListPrint Action.
Example:	<pre>dbpListPrint "AddrBook" "Contacts" "PrintDialog=Yes;Orientation=Landscape;GridLines=Yes;ColumnTitles=Yes; RecordNumbers=Yes;AutoWidth=Yes;UseGridAttributes=Yes;LeftMargin=0.25in; RightMargin=0.25in;TopMargin=0.5in;BottomMargin=0.5in"</pre>
SQL Equivalent:	N/A

dbpListAddHeader

Purpose:	Add a header to appear at the top of a printed list report. You may call this action more than once to create multiple headers. After specifying headers and footers, use dbpListPrint to actually send the report to the printer. You can use dbpListClearHeadersAndFooters to remove headers and footers.
Category:	Simple Reporting
Syntax:	<pre>dbpListAddHeader "database id" "table" "line" "text" "properties"</pre> <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>line</i>

The line number from the top of the page where the header will appear. Subsequent headers (lines 2, 3, etc.) will be printed below the first header (line 1).

text

The text that you want to appear in the header. You may use any combination of text, VisualNEO for Windows variables and the following special codes:

&p Report page number.
 &n Total number of pages in report.
 &t Current time.
 &d Current date.
 && Prints the & symbol.

properties

This is a compound parameter and can contain any combination of the following items:

Alignment= Alignment method used to display the header text. Can be either Left, Right or Center.
 Font=*font* The font name, size, style and character set of the font used for the grid cells. See [Defining Fonts](#).
 FontColor=*color* The color used for the header text. See [Defining Colors](#).

Separate multiple items in a compound variable with semicolons (;).

Example:

The following adds a left and right header to the top of a report:

```
dbpListAddHeader "AddrBook" "Contacts" "1" "Address Book"
"Alignment=Left;Font=Arial,12,Bold,ANSI_CHARSET;FontColor=Black"
dbpListAddHeader "AddrBook" "Contacts" "1" "Page &p of &n"
"Alignment=Right;Font=Arial,10,Normal,ANSI_CHARSET;FontColor=Black"
```

Address Book							Page 1 of 3
#	First Name	Last Name	Company	Street	City	State/Prov	Postal Code
1	Richard	Mendez		199 West Spray	Oceanside	CA	87702
2	Lilly	Blanchet		1662 East Raven St	Salem	MA	68880
3	David	Thompson		128 East 14th St	Breckenridge	CO	87758
4	Seth	Trenton		1588 Williams Court	Williamsburg	VA	37798
5	Rose	Bush		6110 Westchester Ave	Phoenix	AZ	89974
6	Harry	Eastman		12 North Freeson St	Clarkston	ID	98520
7	Violet	Edwards		22 South Eagle Creek Rd	New Bedford	MS	17792
8	Richie	Cunningham		1440 Fifties St	Home Town	CA	90032
9	Deldre	Long		177 Bickford Ave	Warrington	WA	98005
10	Freddy	Fillmore		22 East Palm St	Tampa	FL	29908
11	Ernie	Fernandez		1687 Durango Ct	Dallas	TX	47792
12	Twyla	Thom		936 Saguaro St	Scottsdale	AZ	83356
13	Jeremiah	Johnson		6 Stone's Throw	Grizzly Creek	WY	79982
14	Enrico	Fettucini		24 Canal Rd	Little Venice	NJ	10043
15	Sally	Ride		1959 Cruise Way	Mustang	NV	75589
16	Leroy	Baggins		17225 Mole Hill	Wizard Dale	VT	29985
17	Sue Ellen	Beaumont		922 South Montgomery Ln	Tarrah	SC	32296
18	Teresa	Fenwick		1884 Birdwatch Circle	Bellingham	IN	68897
19	Hector	Eglesias		2244 Bright Lights Ave	Las Vegas	NV	84497
20	Truman	Chipotle		4948 Park Ave	New York	NY	27709
21	Mary	Marcos		190 Selby St	Farmingdale	KS	59980

SQL
Equivalent:

N/A

dbpListAddFooter

Purpose:

Add a footer to appear at the bottom of a printed list report. You may call this action more than once to create multiple footers. After specifying headers and footers, use [dbpListPrint](#) to actually send the report to the printer. You can use [dbpListClearHeadersAndFooters](#) to remove headers and footers.

Category:	Simple Reporting
Syntax:	dbpListAddFooter "database id" "table" "line" "text" "properties" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>line</i> The line number from the bottom of the page where the footer will appear. Subsequent footers (lines 2, 3, etc.) will be printed below the first footer (line 1). <i>text</i> The text that you want to appear in the header. You may use any combination of text, VisualNEO for Windows variables and the following special codes: &p Report page number. &n Total number of pages in report. &t Current time. &d Current date. && Prints the & symbol. <i>properties</i> This is a compound parameter and can contain any combination of the following items: Alignment=<i>value</i> Alignment method used to display the header text. Can be either Left, Right or Center. Font=<i>font</i> The font name, size, style and character set of the font used for the grid cells. See Defining Fonts. FontColor=<i>color</i> The color used for the header text. See Defining Colors. Separate multiple items in a compound variable with semicolons (;).
Example:	dbpListAddFooter "AddrBook" "Contacts" "1" "Page &p of &n" "Alignment=Center;Font=Arial,9,Normal,ANSI_CHARSET;FontColor=Black"
SQL Equivalent:	N/A

dbpListClearHeadersAndFooters

Purpose:	Remove headers and footers previously created using dbpListAddHeader and dbpListAddFooter .
Category:	Simple Reporting
Syntax:	dbpListClearHeadersAndFooters "database id" "table" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table.
Example:	dbpListClearHeadersAndFooters "AddrBook" "Contacts"
SQL Equivalent:	N/A

Advanced Reporting

dbpPrintReport

Purpose:	Print an advanced report created with the Report Designer utility. Tables used by the report should be opened, sorted and queried if needed prior to printing. To view a report prior to printing, see dbpPreviewReport .
Category:	Advanced Reporting
Syntax:	dbpPrintReport "database id" "report file" "options" <i>database id</i> The name assigned to the database. This parameter may be left blank when printing reports that do not contain any database elements. <i>report file</i> The name of the file containing the report to be printed. Report files can be created and edited using the Report Designer utility. <i>options</i> This is a compound parameter and can contain any combination of the following items: PrintDialog=<i>Yes/No</i> Yes = display the standard Windows print dialog prior to printing. No = do not display the print dialog - report will be printed immediately. StartPage=<i>value</i> The number of the first page in the report to be printed. Leave this and EndPage blank to print the entire report. EndPage=<i>value</i> The number of the last page in the report to be printed. Leave this and FirstPage blank to print the entire report. Copies=<i>value</i> The number of copies of the report to print. Separate multiple items in a compound variable with semicolons (;). Unneeded items can be omitted.
Example:	dbpPrintReport "AddrBook" "SampleReport.dbr" "PrintDialog=Yes"
SQL Equivalent:	N/A

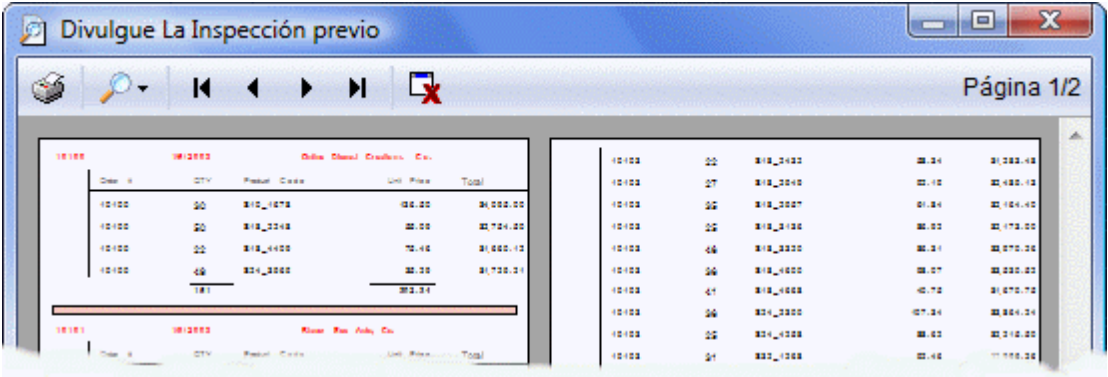
dbpPreviewReport

Purpose:	Preview an optionally print an advanced report created with the Report Designer utility. Tables used by the report should be opened, sorted and queried if needed prior to previewing. The preview screen's interface can be customized using the dbpTranslatePreviewHints action. To print a report without previewing, see dbpPrintReport .
Category:	Advanced Reporting
Syntax:	dbpPreviewReport "database id" "report file" "options" <i>database id</i> The name assigned to the database. This parameter may be left blank when printing reports that do not contain any database elements. <i>report file</i> The name of the file containing the report to be printed. Report files can be created and edited using the Report Designer utility. <i>options</i> This is a compound parameter and can contain any combination of the following items: DisplayMode=<i>value</i> The default preview display mode. Value may be one of the

	following: 200% 150% 100% 75% 50% 25% 10% FitPage FitWidth FacingPages Except for the last three options, the report viewer will display as many pages as will fit on-screen. The number of pages displayed will vary depending on the dimensions of the preview window. InitialPage=value The initial page to display when the preview first appears. The first page (1) is the default.
Example:	dbpPreviewReport "AddrBook" "SampleReport.dbr" "DisplayMode=FacingPages"
SQL Equivalent:	N/A

dbpSetPreviewHints

Purpose:	Customize or translate the hints and text used by dbpPreviewReport's preview screen.																																		
Category:	Advanced Reporting																																		
Syntax:	dbpSetPreviewHints "hints" <i>hints</i> This is a compound parameter and can contain any combination of the following items: <table> <tr> <td>Title=<i>value</i></td><td>Title that appears at the top of the preview window.</td></tr> <tr> <td>Print=<i>value</i></td><td>Print button hint.</td></tr> <tr> <td>Zoom=<i>value</i></td><td>Zoom button hint.</td></tr> <tr> <td>First=<i>value</i></td><td>First page button hint.</td></tr> <tr> <td>Previous=<i>value</i></td><td>Previous page button hint.</td></tr> <tr> <td>Next=<i>value</i></td><td>Next page button hint.</td></tr> <tr> <td>Last=<i>value</i></td><td>Last page button hint.</td></tr> <tr> <td>Close=<i>value</i></td><td>Close button hint.</td></tr> <tr> <td>FitPage=<i>value</i></td><td>Zoom menu's "Fit Page" item.</td></tr> <tr> <td>FitWidth=<i>value</i></td><td>Zoom menu's "Fit Width" item.</td></tr> <tr> <td>FacingPages=<i>value</i></td><td>Zoom menu's "Facing Pages" item.</td></tr> <tr> <td>Page=<i>value</i></td><td>Page number indicator in upper right corner on preview window.</td></tr> <tr> <td>PreparingReport=<i>value</i></td><td>Title that appears at the top of the progress dialog when generating report preview.</td></tr> <tr> <td>Cancel=<i>value</i></td><td>Progress dialog's cancel button.</td></tr> <tr> <td>PerformingFirstPass=<i>value</i></td><td>Progress dialog's first prompt.</td></tr> <tr> <td>PerformingFinalPass=<i>value</i></td><td>Progress dialog's final prompt.</td></tr> <tr> <td>PageXOFY=<i>value</i></td><td>Progress dialog's page number indicator. Within this hint you</td></tr> </table>	Title= <i>value</i>	Title that appears at the top of the preview window.	Print= <i>value</i>	Print button hint.	Zoom= <i>value</i>	Zoom button hint.	First= <i>value</i>	First page button hint.	Previous= <i>value</i>	Previous page button hint.	Next= <i>value</i>	Next page button hint.	Last= <i>value</i>	Last page button hint.	Close= <i>value</i>	Close button hint.	FitPage= <i>value</i>	Zoom menu's "Fit Page" item.	FitWidth= <i>value</i>	Zoom menu's "Fit Width" item.	FacingPages= <i>value</i>	Zoom menu's "Facing Pages" item.	Page= <i>value</i>	Page number indicator in upper right corner on preview window.	PreparingReport= <i>value</i>	Title that appears at the top of the progress dialog when generating report preview.	Cancel= <i>value</i>	Progress dialog's cancel button.	PerformingFirstPass= <i>value</i>	Progress dialog's first prompt.	PerformingFinalPass= <i>value</i>	Progress dialog's final prompt.	PageXOFY= <i>value</i>	Progress dialog's page number indicator. Within this hint you
Title= <i>value</i>	Title that appears at the top of the preview window.																																		
Print= <i>value</i>	Print button hint.																																		
Zoom= <i>value</i>	Zoom button hint.																																		
First= <i>value</i>	First page button hint.																																		
Previous= <i>value</i>	Previous page button hint.																																		
Next= <i>value</i>	Next page button hint.																																		
Last= <i>value</i>	Last page button hint.																																		
Close= <i>value</i>	Close button hint.																																		
FitPage= <i>value</i>	Zoom menu's "Fit Page" item.																																		
FitWidth= <i>value</i>	Zoom menu's "Fit Width" item.																																		
FacingPages= <i>value</i>	Zoom menu's "Facing Pages" item.																																		
Page= <i>value</i>	Page number indicator in upper right corner on preview window.																																		
PreparingReport= <i>value</i>	Title that appears at the top of the progress dialog when generating report preview.																																		
Cancel= <i>value</i>	Progress dialog's cancel button.																																		
PerformingFirstPass= <i>value</i>	Progress dialog's first prompt.																																		
PerformingFinalPass= <i>value</i>	Progress dialog's final prompt.																																		
PageXOFY= <i>value</i>	Progress dialog's page number indicator. Within this hint you																																		

	must include two special codes to indicate where you want the page numbers to appear. Use the code %p to represent the current page number and %t to represent the total number of pages. For example: Página %p de %t Title that appears at the top of the progress dialog when printing a report. Separate multiple items in a compound variable with semicolons (;). Unneeded items can be omitted.
Example:	The following will translate the preview screen's title and page number indicator into Spanish. <pre>dbpSetPreviewHints "Title=Divulgue La Inspección previo;Page=Página"</pre> 
SQL Equivalent:	N/A

Created with the Standard Edition of HelpNDoc: [Easy CHM and documentation editor](#)

Import/Export

dbpImportFromCSV

Purpose:	Import records from an external delimited ASCII file. The format of the ASCII file must be compatible with the table it's being imported into. Files created with the dbpExportToCSV action can be used as well as files created by many commercial database and spreadsheet applications.
Category:	Import/Export
Syntax:	<pre>dbpImportFromCSV "database id" "table" "filename" "properties"</pre> <i>database id</i> The name assigned to the database containing the table you want to import into. <i>table</i> The name of the table that will receive the imported data. <i>filename</i> The name and location of the external file containing the data to be imported. <i>properties</i> This is a compound parameter and can contain any combination of the following items: Delimiter=<i>char</i> The delimiter character used to separate field data in the file. You must specify the correct delimiter or the file will

	not import properly. The most common delimiter character is a comma (,). IncludesFieldNames=Yes/No Yes = The first line in the file contains field names. No = the first line in the file contains data. MatchFieldNames=Yes/No Yes = if possible match the file's field names to the table's field names when importing data. No = ignore the file's field names and import field data into the table in the order it appears. Separate multiple items in a compound variable with semicolons (;).
Example:	dbpImportFromCSV "AddrBook" "Contacts" "[PubDir]data.csv" "Delimiter=,;ContainsFieldNames=Yes;MatchFieldNames=Yes"
SQL Equivalent:	LOAD DATA INFILE <i>filename</i> INTO TABLE <i>table</i>

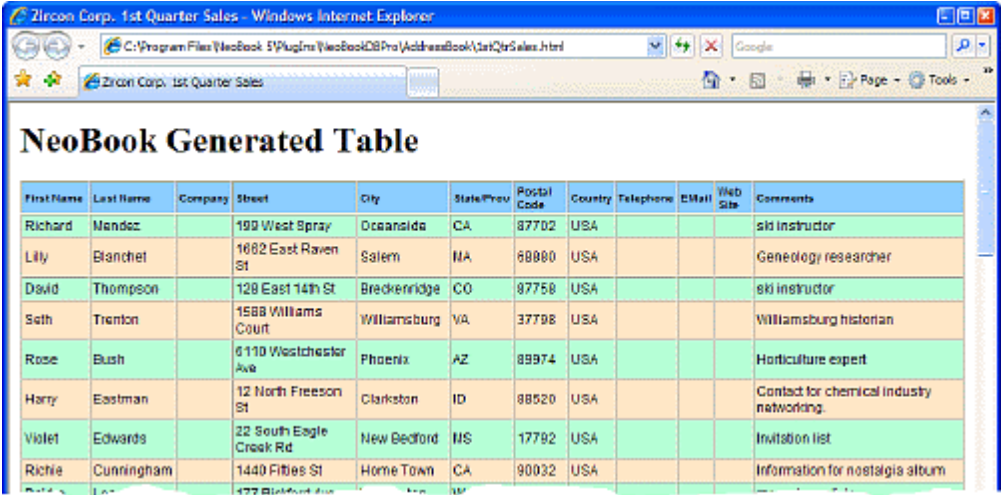
dbpExportToCSV

Purpose:	Export all records matching current search query (see dbpQuery) to an external delimited ASCII text file. If no query is active, the entire table will be exported. Most database and spreadsheet software can import delimited ASCII files. You can modify the format of the exported file using the properties parameter. You can also use dbpSetColumnOrder to limit which fields are exported.
Category:	Import/Export
Syntax:	dbpExportToCSV "database id" "table" "filename" "properties" <i>database id</i> The name assigned to the database containing the table you want to export. <i>table</i> The name of the table to be exported. <i>filename</i> The name and location of the external file where the exported records will be written. <i>properties</i> This is a compound parameter and can contain any combination of the following items: Range=<i>Current/All</i> Use Current to export only the active record or All to export the entire table. Delimiter=<i>char</i> The delimiter character that will be used to separate field data in the file. The most common delimiter character is a comma (,). IncludeHidden=Yes/No Yes = export both visible and hidden fields. No = export only visible fields. Field visibility can be changed using the dbpSetColumnOrder action. By default, all fields are visible. IncludeFieldNames=Yes/No Yes = include the field names at the beginning of the file. No = do not include field names. Field names are used by some commercial database and spreadsheet applications when importing delimited ASCII files. UseColumnTitles=Yes/No For use with IncludeFieldNames property above. Yes = use column titles defined with dbpSetColumnTitles instead of field names. No = use table's actual field names. Append=Yes/No Yes = if the export file already exists append the new data to the end of the existing data. No = overwrite existing export file.

	ForceQuotes=Yes/No Yes = surround all field data with quote characters. No = do not use quote characters unless they are required. Normally quotes are only required when the field data contains the delimiter character. Separate multiple items in a compound variable with semicolons (;).
Example:	dbpExportToCSV "AddrBook" "Contacts" "[PubDir]data.csv" "Range=All;Delimiter=,;IncludeHidden=No;IncludeFieldNames=Yes;Append=No;ForceQuotes=No"
SQL Equivalent:	<pre>SELECT * INTO OUTFILE <i>filename</i> FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' LINES TERMINATED BY '\n' FROM <i>table</i>;</pre>

dbpExportToHTML

Purpose:	Export all records matching current search query (see dbpQuery) to an external HTML document file. If no query is active, the entire table will be exported. The created HTML file will contain a static version of the table at the time of export. You can modify the format of the exported file using the properties parameter. You can also use dbpSetColumnOrder to limit which fields are exported.										
Category:	Import/Export										
Syntax:	dbpExportToHTML "database id" "table" "filename" "properties" <i>database id</i> The name assigned to the database containing the table you want to export. <i>table</i> The name of the table to be exported. <i>filename</i> The name and location of the external HTML file where the exported records will be written. <i>properties</i> This is a compound parameter and can contain any combination of the following items: <table border="0"> <tr> <td>Range=Current/All</td><td>Use Current to export only the active record or All to export the entire table.</td></tr> <tr> <td>IncludeHidden=Yes/No</td><td>Yes = export both visible and hidden fields. No = export only visible fields. Field visibility can be changed using the dbpSetColumnOrder action. By default, all fields are visible.</td></tr> <tr> <td>IncludeFieldNames=Yes/No</td><td>Yes = include the field names at the beginning of the file. No = do not include field names. Field names are used by some commercial database and spreadsheet applications when importing delimited ASCII files.</td></tr> <tr> <td>Title=value</td><td>The HTML page title.</td></tr> <tr> <td>HTMLHeader=value</td><td>A custom header to be used for the HTML document. You may optionally include the following keywords as part of the header:</td></tr> </table> #TITLE This code will be replaced with the value assigned to the Title property above. For example:	Range=Current/All	Use Current to export only the active record or All to export the entire table.	IncludeHidden=Yes/No	Yes = export both visible and hidden fields. No = export only visible fields. Field visibility can be changed using the dbpSetColumnOrder action. By default, all fields are visible.	IncludeFieldNames=Yes/No	Yes = include the field names at the beginning of the file. No = do not include field names. Field names are used by some commercial database and spreadsheet applications when importing delimited ASCII files.	Title=value	The HTML page title.	HTMLHeader=value	A custom header to be used for the HTML document. You may optionally include the following keywords as part of the header:
Range=Current/All	Use Current to export only the active record or All to export the entire table.										
IncludeHidden=Yes/No	Yes = export both visible and hidden fields. No = export only visible fields. Field visibility can be changed using the dbpSetColumnOrder action. By default, all fields are visible.										
IncludeFieldNames=Yes/No	Yes = include the field names at the beginning of the file. No = do not include field names. Field names are used by some commercial database and spreadsheet applications when importing delimited ASCII files.										
Title=value	The HTML page title.										
HTMLHeader=value	A custom header to be used for the HTML document. You may optionally include the following keywords as part of the header:										

	<pre> <head> <title><#TITLE></title> </head> #STYLE This code will be replaced with a cascading style sheet (css) based on the properties assigned to the table grid using the dbpSetGridStyle action. For example, the following will generate a HTML document that closely resembles the formatting used by the table grid: <head> <style type=text/css> #STYLE </style> </head> </pre> HTMLFooter=<i>value</i> A custom footer to be used for the HTML document. Separate multiple items in a compound variable with semicolons (;).
Example:	<pre> SetVar "[Header]" "<html><head><title><#TITLE></title>[#13]<style type=text/css>[#13]#STYLE[#13]</style>[#13]</head><body><p><h1>VisualNEO for Windows Generated Table</h1></p>" SetVar "[Footer]" "
<p>This is a footer!</p>[#13]</body></html>" dbpExportToHTML "AddrBook" "Contacts" "[PubDir]1stQtrSales.html" "Range=All;IncludeHidden=No;IncludeFieldNames=Yes;Title=Zircon Corp. 1st Quarter Sales;HTMLHeader=[Header];HTMLFooter=[Footer]" </pre> 
SQL Equivalent:	N/A

dbpExportToXML

Purpose:	Export all records matching current search query (see dbpQuery) to an external XML file. If no query is active, the entire table will be exported. Many of the newest database and spreadsheet software can import XML files. You can modify the format of the exported file using the properties parameter. You can also use dbpSetColumnOrder to limit which fields are exported.
Category:	Import/Export

Syntax:	dbpExportToXML "database id" "table" "filename" "properties" <i>database id</i> The name assigned to the database containing the table you want to export. <i>table</i> The name of the table to be exported. <i>filename</i> The name and location of the external file where the exported records will be written. <i>properties</i> This is a compound parameter and can contain any combination of the following items: Range=<i>Current/All</i> Use Current to export only the active record or All to export the entire table. IncludeHidden=<i>Yes/No</i> Yes = export both visible and hidden fields. No = export only visible fields. Field visibility can be changed using the dbpSetColumnOrder action. By default, all fields are visible. Separate multiple items in a compound variable with semicolons (;).
Example:	dbpExportToXML "AddrBook" "Contacts" "[PubDir]data.xml" "Range=All;IncludeHidden=No"
SQL Equivalent:	N/A

dbpExportBlob

Purpose:	Export the contents of a picture/blob field from the current record to an external file. The format of the exported file will be identical to the source of the original field data.
Category:	Import/Export
Syntax:	dbpExportBlob "database id" "table" "filename" <i>database id</i> The name assigned to the database containing the table you want to export. <i>table</i> The name of the table containing the field to be exported. <i>filename</i> The name and location of the external file where the picture/blob field data will be written. If the export file already exists, it will be overwritten. Many programs use the three letter file name extension to identify the format of a file. For example, the ".bmp" extension identifies a file as a Windows bitmap (BMP). When inserting images into a picture field, NeoDBpro will keep track of the original file's extension. When using this action to export a picture, you can tell VisualNEO for WindowsDB to use the original file extension by ending the name of the export file with .* instead of a normal extension. For example: MyPhoto.* If the original source of the picture data was a BMP file, NeoDBpro will export the file as: MyPhoto.bmp
Example:	dbpExportBlob "DB1" "Employees" "Photo" "[PubDir]EmpPhoto.jpg"
SQL Equivalent:	N/A

dbpFieldToVar

Purpose:	Copy the contents of a single field from each record to a variable. Items will be separated by the specified delimiter. If a search query is active, only records matching the query will be copied. You can use this action to create a list of items in an entire table. For example, you could extract every part number from a database containing a hardware store's inventory. Those part numbers could then be displayed in one of VisualNEO for Windows's List or Combo Boxes.														
Category:	Import/Export														
Syntax:	dbpFieldToVar "database id" "table" "field" "variable" "properties" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table containing the field to be copied. <i>field</i> The name of the field containing the data to be copied. <i>variable</i> The name of the variable where the field data will be stored. Each item will be separated by the delimiter specified in the properties parameter below. <i>properties</i> This is a compound parameter and can contain any combination of the following items: <table border="0"> <tr> <td>Delimiter=<i>char</i></td><td>The delimiter character that will be used to separate the copied field data. If you plan on displaying the data in a List or Combo box, you should use a carriage return as your delimiter. In VisualNEO for Windows a carriage return is entered as "[#13]".</td></tr> <tr> <td>SkipBlanks=<i>Yes/No</i></td><td>Yes = do not include records where the specified field is empty. No = include blank records.</td></tr> <tr> <td>NoDuplicates=<i>Yes/No</i></td><td>Yes = do not include the same field data more than once. No = allow duplicate data.</td></tr> <tr> <td>DelimiterInContext=<i>value</i></td><td>This option determines what to do when the delimiter character appears within the exported data. Value may be one of the following:</td></tr> </table> <table border="0"> <tr> <td>Leave</td><td>Don't do anything. Leave the data as is.</td></tr> <tr> <td>Replace</td><td>Replace the delimiter with spaces.</td></tr> <tr> <td>UseCSVRules</td><td>Process the data according to the same rules used when exporting to CSV (comma separated values). Data containing the delimiter will be surrounded by double quotes.</td></tr> </table> Separate multiple items in a compound variable with semicolons (;).	Delimiter= <i>char</i>	The delimiter character that will be used to separate the copied field data. If you plan on displaying the data in a List or Combo box, you should use a carriage return as your delimiter. In VisualNEO for Windows a carriage return is entered as "[#13]".	SkipBlanks= <i>Yes/No</i>	Yes = do not include records where the specified field is empty. No = include blank records.	NoDuplicates= <i>Yes/No</i>	Yes = do not include the same field data more than once. No = allow duplicate data.	DelimiterInContext= <i>value</i>	This option determines what to do when the delimiter character appears within the exported data. Value may be one of the following:	Leave	Don't do anything. Leave the data as is.	Replace	Replace the delimiter with spaces.	UseCSVRules	Process the data according to the same rules used when exporting to CSV (comma separated values). Data containing the delimiter will be surrounded by double quotes.
Delimiter= <i>char</i>	The delimiter character that will be used to separate the copied field data. If you plan on displaying the data in a List or Combo box, you should use a carriage return as your delimiter. In VisualNEO for Windows a carriage return is entered as "[#13]".														
SkipBlanks= <i>Yes/No</i>	Yes = do not include records where the specified field is empty. No = include blank records.														
NoDuplicates= <i>Yes/No</i>	Yes = do not include the same field data more than once. No = allow duplicate data.														
DelimiterInContext= <i>value</i>	This option determines what to do when the delimiter character appears within the exported data. Value may be one of the following:														
Leave	Don't do anything. Leave the data as is.														
Replace	Replace the delimiter with spaces.														
UseCSVRules	Process the data according to the same rules used when exporting to CSV (comma separated values). Data containing the delimiter will be surrounded by double quotes.														
Example:	dbpFieldToVar "AddrBook" "Contacts" "LastName" "[Names]" "Delimiter=,;SkipBlanks=Yes;NoDuplicates=Yes"														
SQL Equivalent:	N/A														

dbpRecordToVar

Purpose:	Copy the contents of the current record to a variable. The data from each field will be formatted in delimited ASCII format using the settings specified in the properties parameter. This action is similar to dbpExportToCSV , except that the data is sent to a variable instead of an external file. This action can be used in conjunction with dbpVarToRecord to copy and paste data from one record to another.						
Category:	Import/Export						
Syntax:	dbpRecordToVar "database id" "table" "variable" "properties" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table containing the record to be copied. <i>variable</i> The name of the variable where the record data will be stored. Each field will be separated by the delimiter specified in the properties parameter below. <i>properties</i> This is a compound parameter and can contain any combination of the following items: <table border="0"> <tr> <td>Delimiter=<i>char</i></td><td>The delimiter character that will be used to separate field data in the variable.</td></tr> <tr> <td>IncludeHidden=<i>Yes/No</i></td><td>Yes = export both visible and hidden fields. No = export only visible fields. Field visibility can be changed using the dbpSetColumnOrder action. By default, all fields are visible.</td></tr> <tr> <td>ForceQuotes=<i>Yes/No</i></td><td>Yes = surround all field data with quote characters. No = do not use quote characters unless they are required. Normally quotes are only required when the field data contains the delimiter character.</td></tr> </table> Separate multiple items in a compound variable with semicolons (;).	Delimiter= <i>char</i>	The delimiter character that will be used to separate field data in the variable.	IncludeHidden= <i>Yes/No</i>	Yes = export both visible and hidden fields. No = export only visible fields. Field visibility can be changed using the dbpSetColumnOrder action. By default, all fields are visible.	ForceQuotes= <i>Yes/No</i>	Yes = surround all field data with quote characters. No = do not use quote characters unless they are required. Normally quotes are only required when the field data contains the delimiter character.
Delimiter= <i>char</i>	The delimiter character that will be used to separate field data in the variable.						
IncludeHidden= <i>Yes/No</i>	Yes = export both visible and hidden fields. No = export only visible fields. Field visibility can be changed using the dbpSetColumnOrder action. By default, all fields are visible.						
ForceQuotes= <i>Yes/No</i>	Yes = surround all field data with quote characters. No = do not use quote characters unless they are required. Normally quotes are only required when the field data contains the delimiter character.						
Example:	dbpRecordToVar "AddrBook" "Contacts" "[RecData]" "Delimiter=,;IncludeHidden=No;ForceQuotes=No"						
SQL Equivalent:	N/A						

dbpVarToRecord

Purpose:	Paste the contents of a formatted variable to the current record. The data from each field will be formatted in delimited ASCII format using the settings specified in the properties parameter. This action is similar to dbpImportFromCSV , except that the data originates in a variable instead of an external file. This action can be used in conjunction with dbpRecordToVar to copy and paste data from one record to another.
Category:	Import/Export
Syntax:	dbpVarToRecord "database id" "table" "data" "properties" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table containing the record to be copied. <i>data</i> This can be the actual formatted field data or a variable. The data must match the format of the table otherwise the data could be pasted into the wrong fields. Each field must be separated by the delimiter specified in the properties parameter below.

	<i>properties</i> This is a compound parameter and can contain any combination of the following items: Delimiter=<i>char</i> The delimiter character used to separate field data. IncludeHidden=<i>Yes/No</i> Yes = import data into fields regardless of whether or not they are visible. No = import data only to visible fields. Field visibility can be changed using the dbpSetColumnOrder action. By default, all fields are visible. Separate multiple items in a compound variable with semicolons (;).
Example:	dbpVarToRecord "AddrBook" "Contacts" "[RecData]" "Delimiter=,;IncludeHidden=No"
SQL Equivalent:	N/A

Created with the Standard Edition of HelpNDoc: [Easily create Qt Help files](#)

Stored Procedures

A stored procedure is a set of SQL statements that are stored and executed on the database server. Stored procedures can improve the performance of complex database actions because less information needs to be sent between the server and the client.

Note: *Stored procedures and functions are advanced tools that require familiarity with SQL commands. Not all databases support stored procedures and some, such as MySQL, provide only limited support.*

dbpGetProcedureNames

Purpose:	Obtain the names of all stored procedures in a database and store the results in a variable.
Category:	Stored Procedures
Syntax:	dbpGetProcedureNames "database id" "delimiter" "variable" <i>database id</i> The name assigned to the database. <i>delimiter</i> The character or characters used to separate the procedure names. <i>variable</i> The name of the variable where the procedure names will be stored. Each procedure name will be separated by the specified delimiter.
Example:	dbpGetProcedureNames "DB1" "[#13]" "[ProcList]"
SQL Equivalent:	SHOW PROCEDURES

dbpGetProcedureParameters

Purpose:	Obtain a list of parameters required by a specific stored procedure and store the results in a variable. Note: <i>Many types of databases, including Access and MySQL, do not support this feature and will return an error.</i>
Category:	Stored Procedures

Syntax:	<code>dbpGetProcedureParameters "database id" "procedure" "delimiter" "variable"</code> <i>database id</i> The name assigned to the database. <i>procedure</i> This is the name of the procedure containing the parameters to retrieve. <i>delimiter</i> The character or characters used to separate the parameter names. <i>variable</i> The name of the variable where the parameter names will be stored. Each parameter name will be separated by the specified delimiter.
Example:	<code>dbpGetProcedureParameters "DB1" "SampleProc" "[#13]" "[ParamList]"</code>
SQL Equivalent:	N/A

dbpSetParameter

Purpose:	Assign a value to an input parameter prior to executing a stored procedure. Input parameters are used to pass information to a stored procedure. The number of input parameters and their names depends on the stored procedure. Note: Some databases, such as MySQL, do not correctly initialize stored procedure parameters. When that happens, you must use the dbpAddParameter action to define each of the procedures parameters (both input and output) manually before executing the procedure.
Category:	Stored Procedures
Syntax:	<code>dbpSetParameter "database id" "param name" "input value"</code> <i>database id</i> The name assigned to the database containing the stored procedure. <i>param name</i> The name of the parameter. <i>input value</i> The value to assign to the parameter.
Example:	<code>dbpSetParameter "DB1" "@User" "Bob"</code> <code>dbpSetParameter "DB1" "@Password" "applesauce"</code> <code>dbpExecproc "DB1" "SecurityCheck"</code>
SQL Equivalent:	N/A

dbpAddParameter

Purpose:	Use this action to manually define parameters to be used with a stored procedure. This action is intended to be used with databases, such as MySQL, that do not correctly initialize stored procedure parameters. Each parameter required by the procedure must be defined with a separate call to dbpAddParameter prior to executing the procedure. For most databases that properly initialize stored parameters, you should use the dbpSetParameter action instead.
Category:	Stored Procedures
Syntax:	<code>dbpAddParameter "database id" "param name" "data type" "dir" "input value"</code> <i>database id</i> The name assigned to the database.

	<i>param name</i> The name of the parameter. <i>data type</i> The type of value the parameter represents. Select one of the following: String, Char, Memo, Boolean, Integer, BigInt, AutoInc, Currency, Float, Date, Time, DateTime, Picture <i>dir</i> The type of parameter. Select one of the following: Unknown Parameter type is unknown. Input Parameter can only be used to pass a value to a stored procedure. Output Parameter can only be used to pass a value from a stored procedure back to VisualNEO for Windows. InputOutput Parameter can be used to pass a value both to a stored procedure and back to VisualNEO for Windows. ReturnValue Parameter is used to return a value back to VisualNEO for Windows from a stored function. <i>size</i> Specifies the size of a String or Char type parameters. For all other types, this will be ignored. <i>input value</i> For Input or InputOutput types, this is the value to assign to the parameter. For all other types this will be ignored.
Example:	dbpAddParameter "DB1" "@param1" "Integer" "Output" "" ""
SQL Equivalent:	N/A

dbpClearParameters

Purpose:	Clear all parameters defined with dbpSetParameter or dbpAddParameter . You should use this to clear parameters from memory after running a stored procedure.
Category:	Stored Procedures
Syntax:	dbpClearParameters "database id" "options" <i>database id</i> The name assigned to the database. <i>options</i> This is a compound parameter and can contain any combination of the following items: ClearVariables=Yes/No Yes = clear any VisualNEO for Windows variables associated with stored procedure parameters. No = leave VisualNEO for Windows variables alone.
Example:	dbpClearParameters "DB1" "ClearVariables=Yes"
SQL Equivalent:	N/A

dbpExecProc

Purpose:	Execute a stored procedure or function. Parameters created with the dbpSetParameter
----------	---

	and dbpAddParameter actions are passed to the procedure. After the procedure has executed, NeoDBpro will automatically create variables for each of the output parameters returned by the server.
Category:	Stored Procedures
Syntax:	dbpExecProc "database id" "procedure" "results table" <i>database id</i> The name assigned to the database. <i>procedure</i> This is the name of the procedure to be executed. <i>results table (optional)</i> If the procedure returns a data set, this parameter can be used to enter the name of the table where you want the results of the query to be displayed. This may be the name of an existing database table or it can be a temporary table. If the table specified does not exist in the database, it is assumed to be a temporary table. Leave this parameter blank if the procedure does not return any results. A temporary table exists only while your publication is running and is not physically part of the database. However, most temporary tables derive their contents from real data, so any edits you make may affect other tables. A temporary table created with dbpExecProc can be opened with the dbpOpenTable and displayed with dbpShowGrid actions. Actions that require a physical table, such as dbpAddField or dbpQuery, cannot be used with a temporary table.
Example:	<pre>dbpSetParameter "DB1" "@User" "Bob" dbpSetParameter "DB1" "@Password" "applesauce" dbpExecproc "DB1" "SecurityCheck" ""</pre> The following example executes a stored procedure and sends the results to a temporary table, which is then opened and displayed in a grid: <pre>dbpExecProc "DB1" "CalculateSales" "Temp1" dbpOpenTable "DB1" "Temp1" "" dbpShowGrid "DB1" "Temp1" "Rectangle1"</pre>
SQL Equivalent:	CALL <i>procedure</i> (parameters)

Created with the Standard Edition of HelpNDoc: [Easily create Help documents](#)

Math

dbpSum

Purpose:	Calculate the sum of a single field from each record in a table. If a search query is active, only records matching the query will be included in the calculation. This action could be used to calculate the total value of all items in an inventory or sales table.
Category:	Math
Syntax:	dbpSum "database id" "table" "field" "variable" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>field</i> The name of the field to be used for the calculation. The field must be a compatible

	numeric data type . <i>variable</i> The name of the variable where the calculated value will be stored. The value will be formatted using the field's display format as specified in dbpSetFieldProperties .
Example:	dbpSum "DB1" "Payments" "Amount" "[TotalValue]"
SQL Equivalent:	SELECT Sum(<i>field</i>) FROM <i>table</i>

dbpMin

Purpose:	Calculate the minimum value of a single field from each record in a table. If a search query is active, only records matching the query will be included in the calculation. You could use this action to examine an inventory database to determine the least expensive item in stock.
Category:	Math
Syntax:	dbpMin "database id" "table" "field" "variable" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>field</i> The name of the field to be used for the calculation. The field must be a compatible numeric data type . <i>variable</i> The name of the variable where the calculated value will be stored. The value will be formatted using the field's display format as specified in dbpSetFieldProperties .
Example:	dbpMin "DB1" "Inventory" "Price" "[MinPrice]"
SQL Equivalent:	SELECT Min(<i>field</i>) FROM <i>table</i>

dbpMax

Purpose:	Calculate the maximum value of a single field from each record in a table. If a search query is active, only records matching the query will be included in the calculation. You could use this action to examine an inventory database to determine the most expensive item in stock.
Category:	Math
Syntax:	dbpMax "database id" "table" "field" "variable" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>field</i> The name of the field to be used for the calculation. The field must be a compatible numeric data type . <i>variable</i> The name of the variable where the calculated value will be stored. The value will be formatted using the field's display format as specified in dbpSetFieldProperties .
Example:	dbpMax "DB1" "Inventory" "Price" "[MaxPrice]"

SQL Equivalent:	SELECT Max(<i>field</i>) FROM <i>table</i>
-----------------	--

dbpAvg

Purpose:	Calculate the average value of a single field from each record in a table. If a search query is active, only records matching the query will be included in the calculation. You could use this action to examine an inventory database to determine the average price of all items in stock.
Category:	Math
Syntax:	dbpAvg "database id" "table" "field" "variable" <i>database id</i> The name assigned to the database. <i>table</i> The name of the table. <i>field</i> The name of the field to be used for the calculation. The field must be a compatible numeric data type . <i>variable</i> The name of the variable where the calculated value will be stored. The value will be formatted using the field's display format as specified in dbpSetFieldProperties .
Example:	dbpAvg "DB1" "Inventory" "Price" "[AveragePrice]"
SQL Equivalent:	SELECT Avg(<i>field</i>) FROM <i>table</i>

Created with the Standard Edition of HelpNDoc: [Qt Help documentation made easy](#)

Miscellaneous**dbpShowErrors**

Purpose:	When on, any problems that occur while processing database commands will be displayed to the user in the form of error messages. When off no messages are displayed. When this feature is turned off, you can obtain the most recent error is by examining the global [dbpError] variable. ShowErrors is set to on automatically when NeoDBpro is initialized.
Category:	Miscellaneous
Syntax:	dbpShowErrors "status" <i>status</i> The status of the ShowErrors feature. Use "Yes" to turn ShowErrors on or "No" to turn it off.
Example:	When ShowErrors is turned off you can check the global [dbpError] variable to detect important errors and respond if needed. For example: dbpShowErrors "No" dbpFind "AddrBook" "Contacts" "LastName;FirstName" "[SearchStr]" "ExactMatch=No;CaseSensitive=No" If "[dbpError]" ">" "" AlertBox "Sorry" "No matching records found." EndIf dbpShowErrors "Yes"
SQL	

Equivalent:	N/A
-------------	-----

Note: In addition to the normal error trapping provided by the `[dbpError]` variable, you can also create an error subroutine called **DBPro_OnError** to execute special actions whenever a database error occurs. For example, the following subroutine beeps annoyingly whenever a database error is detected:

```
:DBPro_OnError
    PlayTone "440"
Return
```

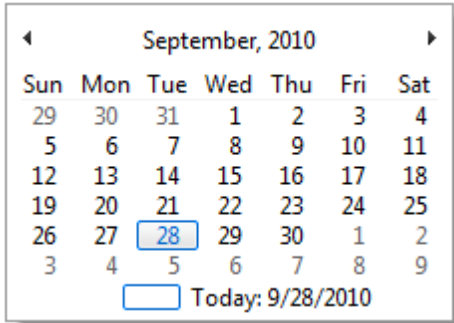
You should avoid executing database actions from within this subroutine if possible! Doing so could cause your publication to become stuck in an endless loop.

dbpTranslateHints

Purpose:	Customize or translate the hints used by the grid navigation buttons and other NeoDBpro features. These settings affects all tables and grids, even those that have not yet been opened. The grid navigation bar is displayed using the ShowNavigationBar property of the dbpSetGridProperties action.																								
Category:	Miscellaneous																								
Syntax:	dbpTranslateHints "hints" <i>hints</i> This is a compound parameter and can contain any combination of the following items: <table> <tr> <td>First=<i>value</i></td><td>value = the hint text for the navigation bar's First button.</td></tr> <tr> <td>Previous=<i>value</i></td><td>value = the hint text for the navigation bar's Previous button.</td></tr> <tr> <td>Next=<i>value</i></td><td>value = the hint text for the navigation bar's Next button.</td></tr> <tr> <td>Last=<i>value</i></td><td>value = the hint text for the navigation bar's Last button.</td></tr> <tr> <td>Add=<i>value</i></td><td>value = the hint text for the navigation bar's Add button.</td></tr> <tr> <td>Delete=<i>value</i></td><td>value = the hint text for the navigation bar's Delete button.</td></tr> <tr> <td>Edit=<i>value</i></td><td>value = the hint text for the navigation bar's Edit button.</td></tr> <tr> <td>Save=<i>value</i></td><td>value = the hint text for the navigation bar's Save Edits or Post button.</td></tr> <tr> <td>Cancel=<i>value</i></td><td>value = the hint text for the navigation bar's Cancel Edit button.</td></tr> <tr> <td>Refresh=<i>value</i></td><td>value = the hint text for the navigation bar's Refresh button.</td></tr> <tr> <td>DeletePrompt=<i>value</i></td><td>value = the message text that appears in the confirmation dialog box that appears when the user attempts to delete a record by pressing the Delete key or clicking the navigation bar's Delete Record button. Set this item to an empty string will disable the delete confirmation.</td></tr> <tr> <td>DeleteTitle=<i>value</i></td><td>value = the caption text for the confirmation dialog box that appears when the user attempts to delete a record by pressing the Delete key or clicking the navigation bar's Delete Record button.</td></tr> </table> Separate multiple items in a compound variable with semicolons (;). Items that do not need to be translated can be omitted.	First= <i>value</i>	value = the hint text for the navigation bar's First button.	Previous= <i>value</i>	value = the hint text for the navigation bar's Previous button.	Next= <i>value</i>	value = the hint text for the navigation bar's Next button.	Last= <i>value</i>	value = the hint text for the navigation bar's Last button.	Add= <i>value</i>	value = the hint text for the navigation bar's Add button.	Delete= <i>value</i>	value = the hint text for the navigation bar's Delete button.	Edit= <i>value</i>	value = the hint text for the navigation bar's Edit button.	Save= <i>value</i>	value = the hint text for the navigation bar's Save Edits or Post button.	Cancel= <i>value</i>	value = the hint text for the navigation bar's Cancel Edit button.	Refresh= <i>value</i>	value = the hint text for the navigation bar's Refresh button.	DeletePrompt= <i>value</i>	value = the message text that appears in the confirmation dialog box that appears when the user attempts to delete a record by pressing the Delete key or clicking the navigation bar's Delete Record button. Set this item to an empty string will disable the delete confirmation.	DeleteTitle= <i>value</i>	value = the caption text for the confirmation dialog box that appears when the user attempts to delete a record by pressing the Delete key or clicking the navigation bar's Delete Record button.
First= <i>value</i>	value = the hint text for the navigation bar's First button.																								
Previous= <i>value</i>	value = the hint text for the navigation bar's Previous button.																								
Next= <i>value</i>	value = the hint text for the navigation bar's Next button.																								
Last= <i>value</i>	value = the hint text for the navigation bar's Last button.																								
Add= <i>value</i>	value = the hint text for the navigation bar's Add button.																								
Delete= <i>value</i>	value = the hint text for the navigation bar's Delete button.																								
Edit= <i>value</i>	value = the hint text for the navigation bar's Edit button.																								
Save= <i>value</i>	value = the hint text for the navigation bar's Save Edits or Post button.																								
Cancel= <i>value</i>	value = the hint text for the navigation bar's Cancel Edit button.																								
Refresh= <i>value</i>	value = the hint text for the navigation bar's Refresh button.																								
DeletePrompt= <i>value</i>	value = the message text that appears in the confirmation dialog box that appears when the user attempts to delete a record by pressing the Delete key or clicking the navigation bar's Delete Record button. Set this item to an empty string will disable the delete confirmation.																								
DeleteTitle= <i>value</i>	value = the caption text for the confirmation dialog box that appears when the user attempts to delete a record by pressing the Delete key or clicking the navigation bar's Delete Record button.																								
Example:	The following example will translate the hints into Italian: <pre>dbpTranslateHints "First=Primo;Previous=Precedente;Next=Successivo; Last=Ultimo;Add=Aggiungi;Delete=Elimina;Edit=Modifica;Save=Salva Modifiche;Cancel=Annulla Modifica;Refresh=Rinfreschi;DeletePrompt=Annotazione di cancellazione?;DeleteTitle=Confermi"</pre>																								

	You can disable the delete confirmation dialog, by setting the DeletePrompt and DeleteTitle properties to empty strings. For example: <pre>dbpTranslateHints "DeletePrompt=;DeleteTitle="</pre>
SQL Equivalent:	N/A

dbpPopupDateSelector

Purpose:	Display a popup calendar and allow the user to select a date. The selected date is stored in a variable. The appearance of the calendar is determined by the current Windows theme.
Category:	Miscellaneous
Syntax:	dbpPopupDateSelector "properties" <i>properties</i> This is a compound parameter and can contain any combination of the following items: Left=<i>number</i> The coordinates of the popup calendar's left corner (in pixels) relative to the publication window. Top=<i>number</i> The coordinates of the popup calendar's top corner (in pixels) relative to the publication window. InitialDate=<i>date</i> The date to be displayed when the calendar first appears. Leave this field blank to display today's date. DateFormat=<i>value</i> The format used for both the InitialDate above and the returned selected date. See Formatting Fields for information on date formats. Separate multiple items in a compound variable with semicolons (;). <i>variable</i> The name of the variable to store the selected date.
Example:	dbpPopupDateSelector "Left=57;Top=301;InitialDate=;DateFormat=m/d/yyyy" "[Result]" 
SQL Equivalent:	N/A

Created with the Standard Edition of HelpNDoc: [What is a Help Authoring tool?](#)

Creating Reports

NeoDBpro provides two types of printed reports - list-based [simple reports](#) and form-based

[advanced reports.](#)

Simple Reports are printed in a tabular list which is similar in appearance to the on-screen grid created with the [dbpShowgrid](#) action. The report is created automatically using the color and font settings applied to the current table grid. Custom headers and footers may be added using the [dbpListAddHeader](#) and [dbpListAddFooter](#) actions. Simple reports are very easy to create, but control over the appearance, formatting and placement of data is limited. Below is an example of a typical simple report:

Address Book							Page 1 of 3
#	First Name	Last Name	Company	Street	City	State/Prov	Postal Code
1	Richard	Mendez		199 West Spray	Oceanside	CA	87702
2	Lilly	Blanchet		1662 East Raven St	Salem	MA	68880
3	David	Thompson		128 East 14th St	Breckenridge	CO	87758
4	Seth	Trenton		1588 Williams Court	Williamsburg	VA	37798
5	Rose	Bush		6110 Westchester Ave	Phoenix	AZ	89974
6	Harry	Eastman		12 North Freeson St	Clarkston	ID	88520
7	Violet	Edwards		22 South Eagle Creek Rd	New Bedford	MS	17792
8	Richie	Cunningham		1440 Fifties St	Home Town	CA	90032
9	Deldre	Long		177 Bickford Ave	Warrington	WA	98005
10	Freddy	Fillmore		22 East Palm St	Tampa	FL	29908
11	Ernie	Fernandez		1687 Durrango Ct	Dallas	TX	47792
12	Twyla	Thom		936 Saguaro St	Scottsdale	AZ	83356
13	Jerimiah	Johnson		6 Stone's Throw	Grizzly Creek	WY	79882
14	Enrico	Fettucini		24 Canal Rd	Little Venice	NJ	10043
15	Sally	Ride		1959 Cruise Way	Mustang	NV	75589
16	Leroy	Baggins		17225 Mole Hill	Wizard Dale	VT	29985
17	Sue Ellen	Beaumont		922 South Montgomery Ln	Tarrah	SC	32296
18	Teresa	Fenwick		1884 Birdwatch Circle	Bellingham	MN	68897
19	Hector	Elesias		2244 Bright Lights Ave	Las Vegas	NV	84497
20	Truman	Chipotle		4948 Park Ave	New York	NY	27709
21	Mary	Marcos		190 Selby St	Farmingdale	KS	59980
22	John	Smith		22 Park Crest Dr	1st of Enslark	DI	18553

Advanced Reports are more difficult to create, but are highly customizable and can display more complicated types of data. Advanced reports can be designed to print mailing labels, lists, invoices, office forms complete with graphics and more. The appearance, formatting and placement of data is entirely customizable. Below is an example of an advanced report:

XTZ Corp. Products

Our Vintage Car models realistically portray automobiles produced from the early 1900s through the 1940s. Materials used include Bakelite, diecast, plastic and wood. Most of the replicas are in the 1:18 and 1:24 scale sizes, which provide the optimum in detail and accuracy. Prices range from \$29.99 up to \$189.99 for some special limited edition replicas. All models include a certificate of authenticity from their manufacturer and come fully assembled and ready for display in the home or office.



The perfect holiday or anniversary gift for executives, clients, friends, and family. These handcrafted model ships are unique, stunning works of art that will be treasured for generations! They come fully assembled and ready for display in the home or office. We guarantee the highest quality, and best value.



Model trains are a rewarding hobby for enthusiasts of all ages. Whether you're looking for collectible wooden trains, electric streetcars or locomotives, you'll find a number of great choices for any budget within this category. The interactive aspect of trains makes toy trains perfect for young children. The wooden train sets are ideal for children under the age of 5.



Unique, diecast airplane and helicopter replicas suitable for collections, as well as home, office or classroom decorations. Models contain stunning

Advanced reports are stored in special database report files (*.dbr) containing the layout, formatting, graphics, etc. that define the report. Report files are created using NeoDBpro's easy-to-use [Report Designer](#) utility. The Report Designer utilizes a VisualNEO for Windows-like interface allowing reports to be created interactively using simple drag and drop commands. Once created, report files can be printed from VisualNEO for Windows using the

[dbpPrintReport](#) or [dbpPreviewReport](#) actions. For information about creating your own reports, see the Report Designer's help file.

Created with the Standard Edition of HelpNDoc: [Easy to use tool to create HTML Help files and Help web sites](#)

License Agreement

The following conditions govern your use of the NeoDBpro plug-in and other materials.

Copyright

©2018 SinLios Soluciones Digitales. All Rights Reserved. This documentation and the software described within are copyrighted with all rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system or translated into any language in any form without the express, written permission of SinLios Soluciones Digitales.

Trademarks

PixelNEO®, VisualNEO for Windows®, VisualNEO for WebApps & Mobile™ and NeoDBpro™ are trademarks or registered trademarks of SinLios Soluciones Digitales. All other brand and product names mentioned in this guide are trademarks or registered trademarks of their respective owners.

License Agreement

The enclosed software, and its accompanying documentation are protected by both United States and international copyright laws. Except as noted below, duplication by any means is strictly forbidden and a violation of copyright laws.

Limited Warranty on Software

SinLios Soluciones Digitales warrants the physical media to be free of defects in materials and workmanship for a period of thirty (30) days from the date of receipt. In the event SinLios Soluciones Digitales receives written notice from you of defects in materials and/or workmanship within the warranty period, SinLios Soluciones Digitales will replace the defective media. Diskettes must be returned to: SinLios Soluciones Digitales; Customer Services; C/ Antonio Machado 7, 28490, Becerril de la Sierra; SPAIN. The remedy for breach of this limited warranty shall be limited to replacement of physical media and shall not include any other damages. The publisher makes no warranty concerning the function or fitness of any programs reproduced on the media included in this package. Neither SinLios Soluciones Digitales, or its authorized agents shall be liable for consequential, special, indirect, or other similar damages or claims, including loss of profits or any other commercial damages, and in no event will the liability for any damages to you or any other person ever exceed the price paid for the license to use the software, regardless of any form of the claim. SinLios Soluciones Digitales specifically disclaims all other warranties, expressed or implied, including, but not limited to, implied warranties of merchantability and fitness for a particular purpose.

NeoDBpro Runtime License

Legal, registered users of VisualNEO for Windows may use NeoDBpro to produce stand-alone executable (EXE) programs containing original publications. These may be distributed as stand-alone executable (EXE) programs, provided your agreement with anyone who will use the stand-alone executable (EXE) programs which you distribute acknowledges the following items:

1. VisualNEO for Windows and NeoDBpro are owned by SinLios Soluciones Digitales;
2. SinLios Soluciones Digitales will not be held responsible for any damages caused by the disks and programs you create and distribute; and
3. SinLios Soluciones Digitales is under no obligation to provide product or technical support to users of the products which you create using VisualNEO for Windows.

You may distribute the compiled stand-alone executable application programs produced by VisualNEO for Windows and NeoDBpro; and report files (*.dbr) created with Report Designer. You may also distribute the ReportDesigner.exe and ReportDesigner documentation with your compiled stand-alone executable application programs provided that: these files are not altered in any way; and it is made clear somewhere your application's documentation that these files are copyrighted by SinLios Soluciones Digitales. You may not distribute serial numbers, product registration cards, or other related materials and printed information which are included with VisualNEO for Windows or NeoDBpro.

