



# Arca Database Xtra<sup>®</sup> Help

[Installation](#)

[Getting Started](#)

[Database Structure](#)

[Inserting, Deleting and Updating Data](#)

[Retrieving Information](#)

[Additional Features](#)

[Xtra Function Reference](#)

[Error Codes](#)

[SQL Support](#)

[Creating Projectors](#)

[How to Order & Register](#)

[Licensing & Availability](#)

[Technical Support](#)

For up-to-date information please visit our web site:  
[xtras.tabuleiro.com](http://xtras.tabuleiro.com)



ARCA DATABASE XTRA HELP: INSTALLATION

The installation procedure is slightly different depending on the version of Director and platform used. Make sure you have administrative rights to create files in the directory where Director is installed on your system.

IMPORTANT: Arca 3.5 is available for Director 11 and 11.5 ONLY. Users of previous versions of Director can follow the instructions below to install Arca 2.x or 3.0.

WINDOWS

Director 11  
/ 11.5

Director  
MX 2004

Director  
MX

Director 8.5

MACINTOSH

Director 11 /  
11.5

Director MX  
2004

Director MX

Director 8.5



ARCA DATABASE XTRA HELP: INSTALLATION: WINDOWS - DIRECTOR 11

INSTALLING THE XTRA ON WINDOWS - Director 11

Decompress the installation .zip file. This will unpack the Xtra, documentation and sample files to a folder named "Arca Database Xtra" on your machine. To install the Xtra, just copy the file Windows\Arca.x32 to the Director 11 XTRAS folder. If your copy of Director 11 is installed at the default location, the Windows Xtra file will be located at:

C:\Program Files\Adobe\Adobe Director 11\Configuration\Xtras\Arca.x32

Now you need to install the files necessary for creation of cross-platform projector for Mac OSX. Go back to the "Arca Database Xtra" directory where the Xtra files were unpacked. Open the **Mac Universal** directory. Now copy the file "Arca Database Xtra.cpio" to the "Configuration\Cross Platform Resources\Macintosh\Xtras" directory used by Director 11. In a default installation of Director this file will end up at the following location:

C:\Program Files\Adobe\Adobe Director 11\Configuration\Cross Platform Resources\Macintosh\Xtras\Arca Database Xtra.cpio

Finally, you need to edit the xtrainfo.txt file to include information about Arca Database Xtra. This information is used by the cross-platform publishing features in Director 11, to locate the files needed when assembling the Mac OSX version of your projector. The xtrainfo.txt file is located by default at:

C:\Program Files\Adobe\Adobe Director 11\Configuration\xtrainfo.txt

Double click the file to open it in notepad, or alternatively edit with any other text editor. You need to add the following line to the end of the file:

## Online Help

[#namePPC:"Arca Database Xtra", #nameW32:"Arca.x32"]

Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.



ARCA DATABASE XTRA HELP: INSTALLATION: WINDOWS - DIRECTOR MX 2004

INSTALLING THE XTRA ON WINDOWS - Director MX 2004

If you have not done so, we recommend updating to Director MX 2004 version 10.1 before installing the Xtra. This will allow creation of projectors for Mac Classic and OSX (Director MX 2004 without the update can only create cross platform projectors for OSX.)

Decompress the installation .zip file. This will unpack the Xtra, documentation and sample files to a folder named "Arca Database Xtra" on your machine. To install the Xtra, just copy the file Windows\Arca.x32 to the Director MX 2004 XTRAS folder. If your copy of Director MX 2004 is installed at the default location, the Windows Xtra file will be located at:

C:\Program Files\Macromedia\Director MX 2004\Configuration\Xtras\Arca.x32

Now you need to install the files necessary for creation of cross-platform projector for Mac OSX. Go back to the "Arca Database Xtra" directory where the Xtra files were unpacked. Open the **Mac Carbon** directory. Now copy the files "Arca Database Xtra.data" and "Arca Database Xtra.rsrc" files to the "Configuration\Cross Platform Resources\Macintosh\Xtras" directory used by Director MX 2004. In a default installation of Director these files will end up at the following locations:

C:\Program Files\Macromedia\Director MX 2004\Configuration\Cross Platform Resources\Macintosh\Xtras\Arca Database Xtra.data

C:\Program Files\Macromedia\Director MX 2004\Configuration\Cross Platform Resources\Macintosh\Xtras\Arca Database Xtra.rsrc

If you are running Director MX 2004 10.1, you can also install the files necessary for creation of cross-platform projector for Mac Classic. Again, go back to the "Arca Database Xtra" directory where the Xtra files were unpacked. Open the **Mac Classic** directory. Now copy the files "Arca Database Xtra.data" and "Arca Database Xtra.rsrc" files to the "Configuration\Cross Platform Resources\Classic\Xtras" directory used by Director MX 2004. In a default installation of Director these files will end up at the following locations:

## Online Help

C:\Program Files\Macromedia\Director MX 2004\Configuration\Cross Platform Resources\Classic\Xtras\Arca Database Xtra.data

C:\Program Files\Macromedia\Director MX 2004\Configuration\Cross Platform Resources\Classic\Xtras\Arca Database Xtra.rsrc

Finally, you need to edit the xtrainfo.txt file to include information about Arca Database Xtra. This information is used by the cross-platform publishing features in Director MX 2004, to locate the files needed when assembling the OSX and Classic versions of your projector. The xtrainfo.txt file is located by default at:

C:\Program Files\Macromedia\Director MX 2004\Configuration\xtrainfo.txt

Double click the file to open it in notepad, or alternatively edit with any other text editor. You need to add the following line to the end of the file:

```
[#namePPC:"Arca Database Xtra", #nameW32:"Arca.x32"]
```

Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.



ARCA DATABASE XTRA HELP: INSTALLATION: WINDOWS - DIRECTOR MX AND 8.5

INSTALLING THE XTRA ON WINDOWS - Director MX and Director 8.5

Decompress the installation .zip file. This will unpack the Xtra, documentation and sample files to a folder named "Arca Database Xtra" on your machine. To install the Xtra, just copy the file **Windows\Arca.x32** to the Director 8.5 or Director MX XTRAS folder. If you have previously installed an older copy of the Xtra make sure to remove or replace it.

These are the default locations of the Xtras folder for each application:

Director 8.5- C:\Program Files\Macromedia\Director 8.5\Xtras

Director MX- C:\Program Files\Macromedia\Director MX\Xtras

Finally, you need to edit the xtrainfo.txt file to include information about Arca Database Xtra. This information is used by the Shockwave publishing features in Director. The xtrainfo.txt file is located by default at:

Director 8.5 - C:\Program Files\Macromedia\Director 8.5\xtrainfo.txt

Director MX - C:\Program Files\Macromedia\Director MX\xtrainfo.txt

Double click the file to open it in notepad, or alternatively edit with any other text editor. You need to add the following line to the end of the file:

```
[#namePPC:"Arca Database Xtra", #nameW32:"Arca.x32"]
```

Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button (Director MX).







ARCA DATABASE XTRA HELP: INSTALLATION: MACINTOSH - DIRECTOR 11

INSTALLING THE XTRA ON MAC OSX - Director 11

Double-click the installation .dmg file. This will mount a disk named "Arca Database Xtra" on your desktop.

The first step is to copy the Universal binary version of the Xtra, which will be used in the authoring environment and also when creating Mac OSX projectors, for both Intel and PPC machines. This file is located in the install disk image, at:

Arca Database Xtra/Mac Universal/Arca Database Xtra.xtra

This file needs to be copied to the Director 11 Xtras folder. The final pathname for the Xtra in a default installation of Director 11 will be:

OSX Volume Name/Applications/Adobe Director 11/Configuration/Xtras/Arca Database Xtra.xtra

Windows projectors can also be created directly on Director 11 running on Mac OSX after installation of the Windows version of the Xtra. It is located on the install disk, at:

Arca Database Xtra/Windows/Arca.x32

Copy this file to the Cross Platform resources directory in Director 11, so that it will be available at:

OSX Volume Name/Applications/Adobe Director 11/Configuration/Cross Platform Resources/Windows/Xtras/Arca.x32

## Online Help

Finally, you need to edit the `xtrainfo.txt` file to include information about Arca Database Xtra. This information is used by the cross-platform publishing features in Director 11, to locate the files needed when assembling the Windows version of your projector. The `xtrainfo.txt` file is located by default at:

OSX Volume Name/Applications/Adobe Director 11/Configuration/xtrainfo.txt

Double click the file to open it in TextEdit, or alternatively edit with another text editor. Make sure to save the file in plain text format, though. You need to add the following line to the end of the file:

```
[#namePPC:"Arca Dabase Xtra", #nameW32:"Arca.x32"]
```

Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.



ARCA DATABASE XTRA HELP: INSTALLATION: MACINTOSH - DIRECTOR MX 2004

INSTALLING THE XTRA ON MAC OSX - Director MX 2004

Double-click the installation .dmg file. This will mount a disk named "Arca Database Xtra" on your desktop.

The first step is to copy the OSX version of the Xtra, which will be used in the authoring environment and also when creating OSX projectors. This file is located in the install disk image, at:

Arca Database Xtra/Mac Carbon/Arca Database Xtra

This file needs to be copied to the Director MX 2004 Xtras folder. The final pathname for the OSX Xtra in a default installation of Director MX will be:

OSX Volume Name/Applications/Macromedia Director MX 2004/Configuration/Xtras/Arca Database Xtra

Director MX 2004 running on Mac OSX can also be used to create Classic projectors, for Mac OS versions 8 and 9. In order to enable this feature you need to copy the Classic version of Arca Database Xtra to the correct location in your Director MX installation. First locate the Classic version of Arca Database Xtra in the install disk:

Arca Database Xtra/Mac Classic/Arca Database Xtra

This file needs to be copied to the following location in the Director MX 2004 folder, to be used for cross-platform publishing. Copy it to:

OSX Volume Name/Applications/Macromedia Director MX 2004/Configuration/Cross Platform Resources/Classic MacOS/Xtras/Arca

Database Xtra

Windows projectors can also be created directly on Director MX 2004 running on Mac OSX after installation of the Windows version of the Xtra. It is located on the install disk, at:

Arca Database Xtra/Windows/Arca.x32

Copy this file to the Cross Platform resources directory in Director MX 2004, so that it will be available at:

OSX Volume Name/Applications/Macromedia Director MX  
2004/Configuration/Cross Platform Resources/Windows/Xtras/Arca.x32

Finally, you need to edit the xtrainfo.txt file to include information about Arca Database Xtra. This information is used by the Shockwave and cross-platform publishing features in Director MX 2004, to locate the files needed when assembling the Classic MacOS and Windows versions of your projector. The xtrainfo.txt file is located by default at:

OSX Volume Name/Applications/Macromedia Director MX  
2004/Configuration/xtrainfo.txt

Double click the file to open it in TextEdit, or alternatively edit with another text editor. Make sure to save the file in plain text format, though. You need to add the following line to the end of the file:

```
[#namePPC:"Arca Database Xtra", #nameW32:"Arca.x32"]
```

Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.





ARCA DATABASE XTRA HELP: INSTALLATION: MACINTOSH - DIRECTOR MX

INSTALLING THE XTRA ON MAC OSX - Director MX

Double-click the installation .dmg file. This will mount a disk named "Arca Database Xtra" on your desktop.

The first step is to copy the OSX version of the Xtra, which will be used in the authoring environment and also when creating OSX projectors. This file is located in the install disk image, at:

Arca Database Xtra/Mac Carbon/Arca Database Xtra

This file needs to be copied to the Director MX Xtras folder. The final pathname for the OSX Xtra in a default installation of Director MX will be:

OSX Volume Name/Applications/Macromedia Director MX/Xtras/Arca Database Xtra

Director MX running on Mac OSX can also be used to create Classic projectors, for Mac OS versions 8 and 9. In order to enable this feature you need to copy the Classic version of Arca Database Xtra to the correct location in your Director MX installation. First locate the Classic version of Arca Database Xtra in the install disk:

Arca Database Xtra/Mac Classic/Arca Database Xtra

This file needs to be copied to the following location in the Director MX folder:

OSX Volume Name/Applications/Macromedia Director MX/Classic MacOS/Xtras/Arca Database Xtra

## Online Help

Finally, you need to edit the xtrainfo.txt file to include information about Arca Database Xtra. This information is used by the cross-platform publishing features in Director MX 2004, to locate the files needed when assembling the Classic MacOS version of your projector. The xtrainfo.txt file is located by default at:

OSX Volume Name/Applications/Macromedia Director MX/xtrainfo.txt

Double click the file to open it in TextEdit, or alternatively edit with another text editor. Make sure to save the file in plain text format, though. You need to add the following line to the end of the file:

```
[#namePPC:"Arca Database Xtra", #nameW32:"Arca.x32"]
```

Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.



ARCA DATABASE XTRA HELP: INSTALLATION: MACINTOSH - DIRECTOR 8.5

INSTALLING THE XTRA ON MAC OS 8 AND 9 - Director 8.5

Running under OSX, double-click the installation .dmg file. This will mount a disk named "Arca Database Xtra" on your desktop. To install the Xtra just copy the file "Arca Database Xtra" from the Mac Classic folder to the Xtras folder of your Director 8.5 installation. The final pathname for the Xtra will be for example:

Macintosh HD:OS9 Applications:Macromedia Director 8.5:Xtras:Arca Database Xtra

Finally, you need to edit the xtrainfo.txt file to include information about Arca Database Xtra. This information is used by the Shockwave publishing features in Director. The xtrainfo.txt file is located in the directory where Director 8.5 was installed, for example at:

Macintosh HD:OS9 Applications:Macromedia Director 8.5:xtrainfo.txt

Double click the file to open it in SimpleText, or another editor capable of saving plain text files. You need to add the following line to the end of the file:

```
[#namePPC:"Arca Database Xtra", #nameW32:"Arca.x32"]
```

Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window.





## ARCA DATABASE XTRA HELP: GETTING STARTED

Arca Database Xtra is a Scripting Xtra. Scripting Xtras are used to extend the Lingo language with new functions and datatypes. Unlike Asset Xtras there is no visual representation of a scripting Xtra in the Director interface, and you can not create castmembers or sprites. The creation of Scripting Xtras is done in your Lingo or JavaScript syntax scripts, by using the #new Lingo keyword.

The first step is to download and install the Arca Database Xtra, following the instructions in the [installation](#) page. Now that Arca is installed, let's verify that the installation was successful. If you are using DirectorMX you should see the Arca entry in the Scripting Xtras context menu, appearing at the top of the message window. Selecting the ARCA submenu and the "put interface" entry will output a list of all commands understood by Arca in the message window. You can also use the following command

Lingo:

```
put the xtralist
```

JavaScript syntax:

```
trace(_player.xtraList)
```

to verify which Xtras are installed, including the version number for each one.

We will now walk through a simple Lingo session using Arca and the message window. It is not necessary to actually try it in Director right now since you probably do not have this database in your system. For now, just try to read the script and understand what it is supposed to be doing:

Lingo:

```
global gDB
```

```
gDB = new (xtra "arca")
```

```
gDB.openDB(the moviepath & "mydatabase")
```

JavaScript syntax:

```
_global.gDB = new xtra("arca")
```

## Online Help

```
_global.gDB.openDB(_movie.path + "mydatabase")
```

This script creates a new instance of the Arca Xtra, stores it in a global object called `gDB`, and opens a previously created database file named "mydatabase", located in the same directory of your current movie. The database file is not encrypted, so we only need to pass one parameter to the `openDB` function, the name of the database file.

Suppose you know that the database file has one table called "users", containing the fields "name" and "age". The following command will retrieve a list with the names of all users which are 18 years old or younger:

Lingo:

```
myresult = gDB.executeSQL("SELECT name, age FROM users WHERE  
age<=18")
```

```
put myresult
```

JavaScript syntax :

```
var myresult = _global.gDB.executeSQL("SELECT name, age FROM users  
WHERE age<=18")
```

```
trace(myresult)
```

The result will be a property list, containing the information you have requested from the database. A sample output would look like:

```
[#rowschanged: 0, #columns: ["name", "age"], #columnType: ["text", "integer"],  
#rows: [{"John", 15}, {"Mary", 13}, {"Paulo", 10}], #errorMsg: 0]
```

Almost all Arca commands produce a property list as an output. You should always check the `#errorMessage` field for possible errors: 0 indicates that there was no error. A function called `explainError()` is available to translate numeric error codes into text messages, and we will cover it in the next pages.

The records matching your search criteria can be found using direct access to the result property list. You can use something like:

Lingo:

## Online Help

`put myResult.rows.count -- to output the number of records`

`put myResult.rows[1] -- to get the first record`

`put myResult.rows[1][2] -- to get the second field (column) of the first record`

JavaScript syntax :

`trace (myResult.rows.count ) -- to output the number of records`

`trace (myResult.rows[1]) // to get the first record`

`trace ( myResult.rows[1][2]) //to get the second field (column) of the first record`

When you have finished working with the database you should close the database file. The Xtra will automatically close the database for you if a problem occurs, if the Xtra object is destroyed or when the application exits as well, but it is good practice to close the database file in your code:

Lingo:

`gDB.closeDB()`

JavaScript syntax :

`_global.gDB.closeDB()`

Finally, you should dispose of the Xtra object:

Lingo:

`gDB = 0`

JavaScript syntax :

`_global.gDB = 0`

That's it. This sample script summarizes the basic process of interfacing with the Arca database, and retrieving data from it. Of course this only covers the basic functions of the Xtra: there are more powerful features like the ability to create selections and return rows or columns individually, as well as insert or update data and even create or define completely new databases. If you are in a hurry please consult the XTRA FUNCTIONS page for more information about specific commands.

## Online Help

You may have noticed that we have actually used only one function of the Xtra to interact with the database and retrieve data: the `executeSQL()` function. This function is used for almost anything in Arca, including querying the database, creating new records, creating new tables, deleting or updating records. There is one reason for this: Arca uses standard SQL queries for almost all data manipulation. SQL is the default query language used by almost all standard database engines, including Microsoft SQL Server, MySQL, PostgreSQL, DB2 and Oracle. A minimal knowledge of SQL is required in order to use Arca successfully. But don't worry, as we will cover the basic SQL syntax required to use Arca in the next pages. Learning a little bit of SQL will be very useful for any database related work you need to develop in the future, be it with Director or any other language or technology, including dynamic web sites.


Now that you have an idea about how Arca is used let's examine the [basic structure](#) of an Arca Database.



## ARCA DATABASE XTRA HELP: DATABASE STRUCTURE

Arca Databases are stored in a single file. The two main structures of a database are TABLES and INDEXES.

Tables are the structures that store your data in the database. Each table is composed of a number of FIELDS, also known as COLUMNS in some database engines. Suppose you want to store information about the users of your application: you could create a table named USERS, with the following fields: NAME, AGE and ADDRESS. Information about each user would be stored in what we call ROWS or RECORDS in the database. Your database table could be represented by the image below, containing records for 3 users:

Table:  

	name	age	address
1	john	34	67 Lib Street, 7888
2	mary	21	878 San Juan Ave
3	paul	60	80 Circle Square

It is important to learn that each field in the database has a TYPE associated with it. Common types are TEXT, NUMERIC and BLOB. Some database engines are very strict about the type of data that can be stored in a field, even limiting the number of characters by using types as CHAR[20]. Arca however uses a loosely typeless database engine, so the type information is only used as a hint about the contents of each field, to aid the conversion to Lingo types when you retrieve your data. It is entirely possible to store any kind of data supported by Arca in any field of the database, regardless of the declared TYPE for each field. You can even create your own types, or leave the type property blank when creating your tables. This is a feature inherited from the SQLite engine used in Arca. We however recommend that you use the TEXT type to store Lingo strings, the NUMERIC type to store floats and integers, and the BLOB type to store media and pictures (binary data.) This will make it easier for other developers working with your database to identify immediately which type of data was intended to be stored on each field, and it will give the Xtra a hint on how to convert the data back to Lingo. This can be important if you want to interpret numeric data as a string. Take for example the value "20.67". If you store this value on a field of type NUMERIC the Xtra will assume that you want to interpret it as a numeric Lingo value, and will return the float 20.67 in the results. If however you store the same information in a field of type TEXT the Xtra will return this result as a Lingo string, "20.67".

Field names and types are defined when you create a table. But in order to create a

## Online Help

table you must first create the database file, of course. Database files can be created with the `createdb` function in the Arca Xtra, or the Arca Database Browser application. There is no need to specify a specific 3 letter extension for the database filename, but you may specify one if you want to. The following script will create an unencrypted database file named "mydata" in the same directory of the current Director movie:

Lingo:

```
gDB = new (xtra "arca")  
  
gDB.createDB(the moviepath & "mydata")
```

JavaScript syntax :

```
gDB = new xtra ("arca")  
  
gDB.createDB(_movie.path + "mydata")
```

Arca 2 introduces support for encrypted database files. Files can be encrypted or decrypted at any time using the `changekey` function, but you can also specify an optional encryption key when creating the database:

Lingo:

```
gDB = new (xtra "arca")  
  
gDB.createDB(the moviepath & "mydata", "mysecretkey")
```

JavaScript syntax :

```
gDB = new xtra ("arca")  
  
gDB.createDB(_movie.path + "mydata", "mysecretkey")
```

Databases are empty when they are created: they have no tables with data stored, or indexes. It is possible to create tables and indexes using Lingo as well, but this requires knowledge of the SQL language. It is generally much easier to use the Arca Database Browser application to create your database and design the tables and indexes, however. For reference, you can create the users table described above

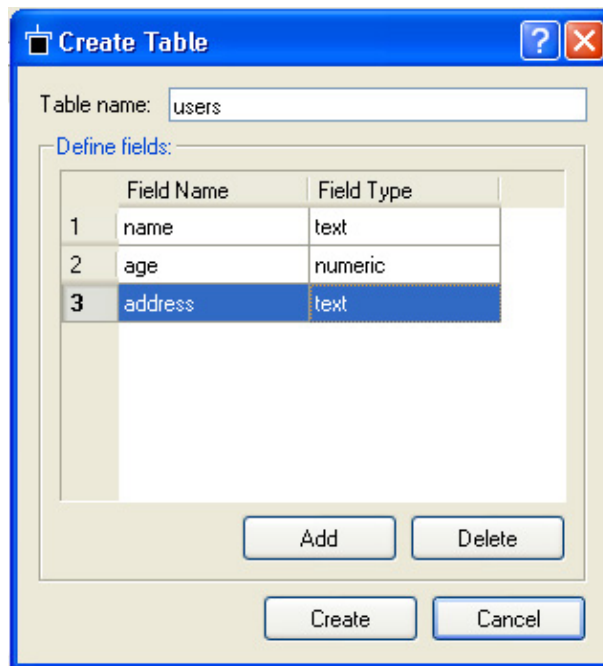
## Online Help

using Lingo or JavaScript syntax with the following command:

```
gDB.executeSQL("CREATE TABLE users(name text, age numeric, address text)")
```

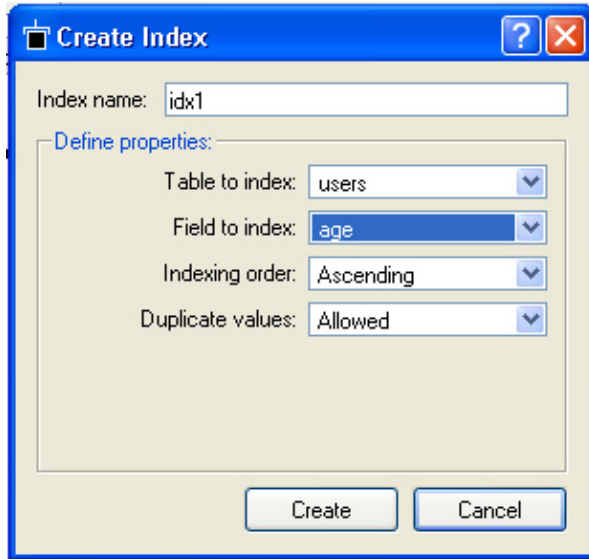
We however recommend using the Arca Database Browser application, as it provides wizards that help you define the table fields and types before creation. If you have not downloaded it yet, please get the Arca Database Browser from the **DOWNLOAD** section at <http://xtras.tabuleiro.com>. All screenshots of data contained in this guide were taken with it. The database browser tool is available in versions for Windows, MacOSX and Linux.

Creating a database with the Arca Database Browser is simple: just select the **CREATE DATABASE** menu entry. Choose a name and location for your database file. The browser will open the **CREATE TABLE** wizard, where you can define the name of your table, and click the **ADD** button to define fields and types.



The Arca Database Browser also lets you create indexes, with the **CREATE INDEX** button. Indexes do not store data, and you do not use them directly. They are used internally by the database engine to speed up certain search operations. In order to create an index you have to define the table and the field to be indexed, and the indexing order (Ascending or Descending). Indexes can also be **UNIQUE**, and in this case the indexed field does not allow duplicate data to be inserted in

different records or rows (for example you could not have two employees with the same userid value if the userid field is being indexed as UNIQUE.) The figure below shows the Create Index window in the Arca Database tool:



Of course, you can create indexes using scripting as well. Remember that almost all operations in Arca can be done using the executeSQL command. The syntax to create the index above would look like this:

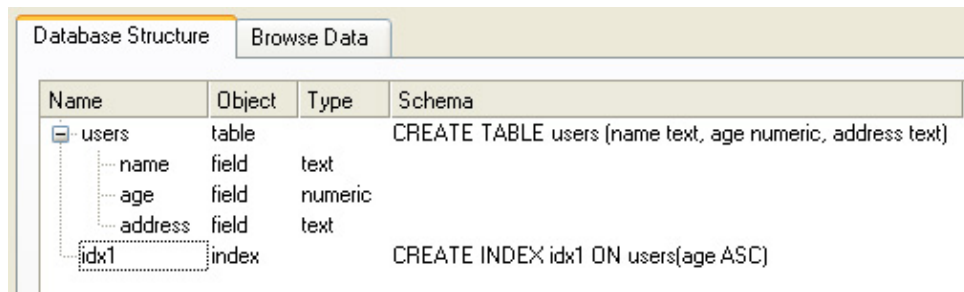
```
gDB.executeSQL("CREATE INDEX idx1 ON users(age ASC)")
```

For most Director applications however indexes are really not necessary: Arca is very fast and optimized, and it already maintains an internal index using a special property called ROWID (more on this later.) You may need to create an index however if your table contains lots of rows (10,000 for example), and you always search for data using an expression that will benefit from an index. In our example it would make sense to index the table using the AGE field if your application usually tries to find users that are older than 18, or present a list of users sorted by age.

A tip: the Arca Database Browser application also lets you see the database structure, including all tables and indexes, and also the SQL commands that were used internally to create them, as you can see in the figure below. You can study the SQL commands used (also called the schema of the database) if you want to recreate tables or indexes later using Lingo, or to learn a bit more of SQL.



## Online Help



The screenshot shows a window with two tabs: 'Database Structure' (selected) and 'Browse Data'. Below the tabs is a table with columns 'Name', 'Object', 'Type', and 'Schema'. The table lists the 'users' table and its fields ('name', 'age', 'address') and an index 'idx1' on the 'age' field.

Name	Object	Type	Schema
users	table		CREATE TABLE users (name text, age numeric, address text)
name	field	text	
age	field	numeric	
address	field	text	
idx1	index		CREATE INDEX idx1 ON users(age ASC)

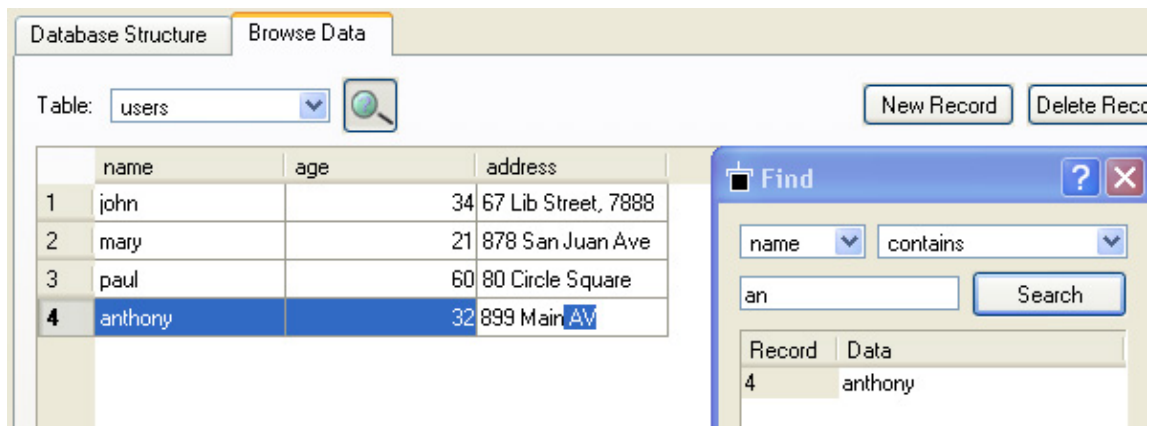
An Arca database may have several tables and indexes, depending on your needs. All data is stored in the same database file, no matter how many tables you use. Let's now check how to insert, delete and update data in your databases.



**ARCA DATABASE XTRA HELP: INSERTING, DELETING AND UPDATING RECORDS**

Records or rows are the places in your database tables where you store actual data. Each row has all the fields defined in the table, and each field contains a single value as we saw previously. Values stored in an Arca database are automatically translated to the Lingo types supported: string, integer, float, picture, media and image objects, and ByteArrays in Director 11.5 as well.

You can use the Arca Database Browser application to add data to your database using a familiar spreadsheet-like interface. There is no need to write functions or Lingo code to import your data, unless you want to do so. Click the **BROWSE DATA** tab in the application, and use the **ADD RECORD** button to create a new, empty record. You can type data directly in the fields, or use copy and paste shortcuts. There are also commands to delete records, and a **FIND** button that you can use to locate information on large databases. If you need more help using the Arca Database Browser application try selecting the **HELP->WHAT'S THIS?** menu item, and clicking on a button or control in the application. The figure below shows a minimized **BROWSE DATA** view in the middle of a data editing session.



The Arca Database Browser is an ideal solution to insert, delete and update strings and numeric data, but it can also be used to import images from PNG or JPEG files. PNG files are converted to image objects, and retain their alpha channel, if present. JPEG files are imported as media values wrapping the original JPEG file, with the original compression intact. The browser application is also only useful when you are preparing a database to be used in your Director projector, and you can even redistribute it freely to your customers if they have a need to edit or consult the database files you have generated. However in most cases you will want to insert and update records using Lingo, directly from your Director application. In this case you have to use the Lingo functions available in the Arca Xtra.

The Arca Xtra uses standard SQL commands and the executeSQL function to insert data in your databases. The syntax for this uses the standard SQL command INSERT. For example to insert information about a new user in our sample table we could use the following SQL command:

```
gDB.executeSQL("INSERT INTO users VALUES('john', 33, '178 Bond Street')")
```

Notice how you always have to quote string values, or the command will generate a SQL error. That's the reason why you need to use 'john' and not simply john in your SQL command string. You will of course notice that this can quickly become a problem: what if you want to insert the string 'john's "good old" house' in the database? This value has both the ' and the " characters, and it would cause Lingo errors if you enter it directly in the executeSQL command. The standard way to deal with this problem in other SQL databases is called "escaping". You use a special character (\) to mark the characters to be escaped. In this case, the standard SQL way to insert this string would be to modify it to 'john\'s \'good old\' house'. You can do this if you want, but it is more complicated than what it needs to be.

Fortunately Arca provides a way for you to enter values without worrying about quoting and escaping. Just mark with question marks the positions in the SQL string where your values need to appear, and include the values in a Lingo list following the SQL query string. In our case, you could write:

Lingo:

```
gDB.executeSQL("INSERT INTO users VALUES(?,?,?)", ["john", 33, "178 Bond Street"])
```

JavaScript syntax :

```
gDB.executeSQL("INSERT INTO users VALUES(?,?,?)", list( "john", 33, "178 Bond Street"))
```

In most cases the values you are going to insert or update are coming from a text or field member, or a scripting variable. This method allows you to use these values directly without worrying about special characters, as any of the acceptable Director value types (string, float, integer, media, picture, image) can be included in the list. For example:

Lingo:

```
gDB.executeSQL("INSERT INTO users VALUES(?,?,?)", [member(1).text, gMyAge, sprite(2).member.image])
```

JavaScript syntax:

```
gDB.executeSQL("INSERT INTO users VALUES(?,?,?)", list(member(1).text, gMyAge, sprite(2).member.picture))
```

Arca will verify, convert and quote/escape the values automatically for you, so there is no need to worry about special characters not supported in SQL commands. It will even compress the

## Online Help

media and picture data automatically for you, using lossless compression algorithms in order to reduce the size of your database!

The same syntax is valid for the SQL UPDATE command, which is used to update data in a table. One example:

Lingo:

```
gDB.executeSQL("UPDATE users SET age=? WHERE name=?", [gNewAge, gUserName])
```

JavaScript syntax :

```
gDB.executeSQL("UPDATE users SET age=? WHERE name=?", list(_global.gNewAge, _global.gUserName))
```

The command above will look for records with the NAME field matching the value of the global gUserName script variable, and will update the AGE field of each one to the value of gNewAge.

A tip: this technique of using a Director list and '?' characters inside the SQL string can be used for any SQL command you use with Arca, both in the executeSQL and createSelection Arca commands, as you can see in the example below:

Lingo:

```
minimumAge = 18
```

```
myselection = gDB.createSelection("SELECT * from users WHERE age>?", [minimumAge])
```

JavaScript syntax :

```
var minimumAge = 18
```

```
var myselection = gDB.createSelection("SELECT * from users WHERE age>?", list(minimumAge))
```

Finally you can also use SQL commands to delete records. A sample line would be something like:

Lingo:

```
gDB.executeSQL("DELETE FROM users WHERE name LIKE ?", ["ma%"])
```

JavaScript syntax:

## Online Help

```
gDB.executeSQL("DELETE FROM users WHERE name LIKE ?", list( "ma% "))
```

This command uses the SQL 'LIKE' operator to delete all records in the user table where the name starts with the letters 'ma'. This will delete the records of users named "mark" and "MARCUS" for example, as well as all other records in the table that match the criteria following the WHERE keyword. If you want to delete all records in a table you can use something like:

```
gDB.executeSQL("DELETE FROM users")
```

Since there are no operators the query will match all records, and all rows will be removed.

Now that you know how to insert and update values in the Arca Database let's see how you can retrieve information stored in it.



## ARCA DATABASE XTRA HELP: RETRIEVING INFORMATION

There are two ways to retrieve rows (or records) from an Arca database file using the Arca Database Xtra. Both use SQL queries to determine which records and fields are going to be retrieved, and present the results as Lingo lists, ready to be used.

### A) Using executeSQL

The first technique uses the standard executeSQL command, and the SQL SELECT keyword. This was explained briefly in the GETTING STARTED page, but let's see it again:

Lingo:

```
myresult = gDB.executeSQL("SELECT name, age FROM users WHERE  
age<=18")
```

```
put myresult
```

JavaScript syntax:

```
var myresult = gDB.executeSQL("SELECT name, age FROM users WHERE  
age<=18")
```

```
trace (myresult)
```

The result will be a property list, containing the information you have requested from the database. A sample output would look like:

```
[#rowschanged: 0, #columns: ["name", "age"], #columnntype: ["text", "integer"],  
#rows: [["John", 15], ["Mary", 13], ["Paulo", 10]], #errorMsg: 0]
```

As you can see, the SELECT...FROM statement allows you to specify the fields and table to be used, and the WHERE operator filters the results. The result of this call is a Lingo property list. Always check the #errorMsg property before you use the other values. Example

Lingo:

```
if myResult.errorMsg<>0 then  
alert("Problem getting data")  
end if
```

JavaScript syntax:

```
if (myResult.errorMsg!=0){  
_player.alert("Problem getting data")  
}
```

You should also pay attention to the other values returned in the result property list. The #rows property will be a list containing the rows retrieved by your query. You can use for example

```
myResult.rows.count
```

to get the number of rows available, and

```
myResult.rows[1]
```

to get the first record retrieved as a Lingo list. You can take it a step further, of course. Suppose you want to retrieve the second field in the third row of the result, simply use

```
myResult.rows[3][2]
```

The #columns property is a list containing the name of the fields retrieved by your query. In the same way, the #columnstype property contains a list with the types of each field retrieved. This is usually not needed but it may be necessary if you are building for example a database browser or some complex front end to a database catalog.

## Online Help

The final element is the #rowsChanged property. In the case of a SELECT command this will always be 0, since SELECT only retrieves data from the database, it does not modify it.

It is possible to retrieve an entire table in this way, with a command like:

```
gDB.executeSQL("SELECT * users")
```

Of course, this has the potential to create a very large list. It is a known fact that Director lists are not very fast, so for situations where you want to retrieve lots of data at the same time you should use a more advanced technique.

### B) Using createSelection

The second method to retrieve data from an Arca database involves creating a selection, which is a collection of results in the database that can be explored using additional functions in the Xtra. This makes sense if your selection is expected to retrieve lots of matches (for example the whole database), and you want to retrieve just one row or a number of rows at a time. This is usually the case when you are browsing a database table, for example: you can instruct the Xtra to create a selection containing the whole table, but only present 10 or 20 records to the user at each page in your application.

You create a selection using a command very similar to the executeSQL technique:

```
myresult = gDB.createSelection("SELECT * FROM users WHERE age>18")
```

Any valid SELECT statement can be used to create a selection. As with the executeSQL method this will also generate a Lingo property list, but with different properties. In the example above, the following could be generated:

```
[#errorMsg: 0, #selectionid: 28556312, #numrows: 2442, #columns: ["name",  
"age", "address"], #columnType: ["text", "numeric", "text"]]
```

You can see that we will get the familiar #errorMsg, #columns and #columnType properties, with the same information found in the executeSQL results. But instead



## Online Help

of a number of rows we will get two additional properties: #numrows and #selectionid.

The #numrows property is simply the number of rows available in your selection. In this case, it shows that 2442 records are available and match the search criteria (age>18). But how do you retrieve this rows? You do this with two additional functions in the Xtra: getRows and getField. In order to use these commands you are required to use the #selectionID property, so that the Xtra knows which selection you are trying to access. You can have several selections opened at the same time in your database.

Suppose you want to retrieve the first 3 rows in this result. You could use the getRows command, which takes the following parameters: selectionID, startingRow, endRow.

Lingo:

```
put gDB.getRows(myResult.selectionid , 1, 3)
```

JavaScript syntax:

```
trace (gDB.getRows(myResult.selectionid , 1, 3))
```

Result:

```
[#columns: ["name", "age", "address"], #columnType: ["text", "numeric", "text"],  
#rows: [["john", 22, "178 Bond Street"], ["mary", 70, "23 Circle Park"],  
["anthony", 31, "4567 Privet Drive"]], #errorMsg: 0]
```

This command will produce a property list containing the same fields used in a normal executeSQL command, with a #rows property containing the requested data. The advantage of using this method is that Lingo only has to generate a #rows list with only 3 elements, instead of your whole selection. This is much faster, of course. In the same way, you can retrieve one unique row of data, using something like

```
myval = gDB.getRows(myResult.selectionid , 300, 300)
```

This will return the record number 300 in your selection in the #rows property of the result.

## Online Help

Another useful technique when working with selections is to retrieve a named field directly. Suppose you want to get the "age" field for the 1256th record in your selection, simply use

```
ageValue = gDB.getField(myResult.selectionid, 1256, "age")
```

Notice that this will return the Director value directly (or Void if the field could not be found), without the need to parse a list. This is the easiest way to get results, but it may not be as fast as getting the rows property directly if you are retrieving large amounts of data at the same time.

Finally, it is important to understand that selections take up memory when you create them. You should always keep track of your selections, and release the memory used by them when you no longer need them. The freeSelection command is used for this:

```
gDB.freeSelection(myResult.selectionid)
```

The Xtra will free the selections for you when you close your database file, but it is good practice to free the selections in your code when you are no longer using them. If you have lost track of your selection objects for some reason (maybe you have not saved a reference to the scripting variable that refers to it) you can use the command

```
gDB.freeSelection(0)
```

to free all selections currently referenced by this database.



## ARCA DATABASE XTRA HELP: ADDITIONAL FEATURES

### The ROWID Field

A very important characteristic of Arca databases is that every table has an invisible field that is always present, the ROWID. This field is created, maintained and indexed automatically by the Arca database engine every time you create a table or add a record to it. The value of the ROWID is a unique number that identifies each row in the table. Remember that you can not set the value of the ROWID: it is set automatically by the database engine when a record is created. This value is not displayed in the Arca Database Browser, but you can always retrieve the value in your SQL commands, and use it as a temporary unique identifier for each row in a database. The Arca Database Browser for example always presents the table data sorted by ROWID, so older records are displayed first and new records are always created as the last record in the spreadsheet.

The tutorial files available in the Arca Database Xtra page use the ROWID extensively to keep track of current records inside a selection. Just keep in mind that the ROWID will change when a database file is compacted, so while useful for temporary identification of records it is not a substitute for a real INTEGER PRIMARY KEY field in more complex applications with multiple tables.

### The INTEGER PRIMARY KEY type

Sometimes it is necessary to have a unique key that identifies each record on a table. The ROWID internal field is an option for temporary data, but it does not survive a compact operation and is also not exported to external .sql dumps, for example. For these cases you can declare a field of type INTEGER PRIMARY KEY in your table. The Arca Database Browser allows creation of INTEGER PRIMARY KEY field types easily, or you can use this type in the CREATE TABLE SQL statement directly.

Arca will automatically insert a unique integer incremental value every time you insert a new record using NULL as the INTEGER PRIMARY KEY field value. In this special case the value of the field can be subsequently retrieved by the getLastInsertRowID() function, for convenience. Here is an example session that you can try using the Arca Database Browser SQL query window, but of course you can issue the same sequence of SQL commands using the Arca Xtra and the executeSQL function:

```
CREATE TABLE mytable (id INTEGER PRIMARY KEY, name text, age
numeric);
```

## Online Help

```
INSERT INTO mytable VALUES(NULL, 'John', 33);
```

```
INSERT INTO mytable VALUES(NULL, 'Mary', 24);
```

If you check the values in the browser window you will see that the ID field has the values 1 and 2 for John and Mary, respectively. You can also use the NEW RECORD button in the browser to add additional records, and they will automatically be created with an incremental value in the id field. The id value in this case is part of the table data, so it will survive a compact operation and even a dump/restore of the table data to an external file.

### COLLATE LOCALE

This is a SQL clause used to retrieve data sorted in the order determined by the system locale. This is used in conjunction with the ORDER BY condition in the SQL query:

```
SELECT firstname, lastname FROM users ORDER BY name COLLATE  
LOCALE;
```

This is useful to display accented characters in the correct order when your program runs on a system set to use the German or French locale, for example. The system runtime strcoll function is used for the comparison.

Compatibility note: the COLLATE LOCALE operator is not functional on MacOS Classic systems and also when using the OSX version of the Arca Database Browser. In these systems no error will be returned, but data will continue to be sorted using the standard C locale. The function works correctly in the Arca Database Browser on Linux and Windows, and is also available in the Arca Database Xtra running under OSX and Windows.

### Database Encryption

## Online Help

Arca 2 supports encryption of database files on disk using a strong algorithm. Data is encrypted and decrypted on the fly and automatically. Encrypted database files can not be read by users without the encryption key, and are not compatible with standard SQLite 3.x database files.

You can create an encrypted database using the Arca Database Browser or by passing an optional encryptionkey string to the createDB Lingo function, documented in the functions list. Once encrypted there is nothing on the database file that identifies it as an Arca database. To open an encrypted database you need to pass the encryption key string as the second parameter of the openDB function. You can also remove, add or change the encryption key for a database file at any time using the changeKey function in the Xtra, or the Arca Database Browser. Again, please consult the [functions reference](#) page for more information. Non-encrypted files can be exchanged with other users or other tools and engines that understand the SQLite 3.x database format.

### Database Encoding

Arca databases by default store string data as Unicode characters, using UTF8 encoding. This makes it possible to interchange data with other database engines that also expect UTF8, including SQLite 3 standard files. Versions of Director older than Director 11 however do not understand Unicode, so the Arca Xtra translates strings automatically for you, depending on the platform it is running. By default Arca uses the #MACROMAN client encoding when translating data to Lingo on the Macintosh platform, and the #LATIN1 client encoding when translating string data on Windows. The default database encoding is UTF8. When running under Director 11 (both MacOSX and Windows) Arca will exchange data directly in UTF8 format, with no translation necessary.

Translation only applies to string variables in Lingo: all other media types are not translated. Also only single byte characters (including accented characters for western languages) are translated reliably on versions of Director before Director 11, as the encoding used by Director for double byte data can not be guessed for Unicode transformation.

For almost all applications there is no need to change the default encoding. However, if you are trying to use Arca to store double byte text in Director MX 2004 and earlier it might be necessary to disable the translation, so string data will stored exactly as passed by Lingo to the Xtra engine. The setEncoding function described in the [functions reference](#) page can be used to change the default translation used by the Xtra in both the Lingo (#CLIENT) or database (#DB) side. Please notice that changing the encoding does not change the actual data on the database file, only its translation to Lingo, and the translation used in future operations.

## Online Help

Note: the Arca Database Browser application also operates with UTF8 encoding as default. This setting can be changed in the PREFERENCES menu if necessary.

Compatibility note: older versions of Arca (pre-2.x) used #LATIN1 as the #DB side encoding, #LATIN1 as the client encoding on Windows and #MACROMAN as the client encoding on the Macintosh. If you are converting older databases that contain accented characters you may want to continue using these settings to avoid having to re-import your data.



## ARCA DATABASE XTRA HELP: XTRA FUNCTIONS

The following is a list of scripting functions available after installation of the Arca Database Xtra.

`arcaregister([1111,2222,3333])` - global function, used to register the Arca Database Xtra. It can be called at any time, usually when the Director movie starts, before your Arca database is opened or created. Unregistered versions of the Xtra are fully functional for evaluation purposes, but they will display a warning the first time a database is opened or created.

Arca serial number are strings, and have the generic format `ARCXX-1111-2222-3333`, where `XX` is the major Xtra version. In order to protect your serial number from being included as a string in your Director projectors or dcr movies, the `arcaregister` function requires only the three groups of numbers 1111, 2222 and 3333 inside a Director list. Leading zeroes do not need to be entered.

An example: if your serial number is `ARC10-0123-4567-0089` then you should register using the following command, usually on a startmovie handler:

Lingo:

```
arcaregister([123, 4567,89])
```

JavaScript syntax :

```
arcaregister(list(123, 4567,89))
```

`new(object me)` - Used to create an instance of the Arca Database Xtra. This is the standard command used in Director to create scripting xtra objects, and returns an instance of the Xtra required to use additional Lingo functions.

Lingo:

```
myDB = new(xtra "arca")
```

```
put myDB
```

```
-- <Xtra child "Arca" 2 239d10>
```

JavaScript syntax :

```
myDB = new xtra("arca")  
  
trace (myDB)  
  
//<<Xtra child "Arca" 2 239d10>>
```

xtrainstance.explainError(integer errorCode) - Used to translate Arca error codes into human readable strings. This command can be used to check and present results of error messages to the user of your application. For a list of error codes please consult the Arca error code table.

Lingo:

```
put myDB.explainError(604)  
  
-- "Can not find the database file"
```

JavaScript syntax :

```
trace (myDB.explainError(604))  
  
// "Can not find the database file"
```

xtrainstance.openDB(string databasefile, optional string encryptionkey) - Open the file specified as a database. If your database file is encrypted you need to supply a second parameter, the encryption key. Returns a Lingo property list, containing any errors generated by the command as the #errorMsg property. Please notice that specifying a wrong encryption key does NOT immediately returns an error, but subsequent attempts to get/set data on the database will produce error message 26, "Database file is encrypted or is not valid". If you want to test for this condition immediately you can use the getDBschema() function after openDB, and check for any errors.

Lingo:

```
put myDB.openDB(the moviepath&"mydbfile")  
  
-- [#errorMsg: 0]  
  
put myDB.openDB(the moviepath&"myencrypteddbfile", "mysecretkey" )  
  
-- [#errorMsg: 0]
```



## Online Help

JavaScript syntax:

```
trace (myDB.openDB(_movie.path + "mydbfile"))  
  
// [#errorMsg: 0]  
  
trace (myDB.openDB(_movie.path + "mydbfile", "mysecretkey"))  
  
// [#errorMsg: 0]
```

xtrinstance.createDB(string databasefile, optional string encryptionkey) - Creates a new, empty database file at the location specified. To create an encrypted database you can supply a second parameter, the encryption key to be used. Returns a Lingo property list, containing any errors generated by the command as the #errorMsg property.

Example:

```
put myDB.createDB("E:\mydocuments\newdatabase")  
  
-- [#errorMsg: 601]  
  
put myDB.explainError(601)  
  
-- "Can not create database file"  
  
put myDB.createDB("C:\mydocuments\newencrypteddatabase", "mysecretkey" )  
  
-- [#errorMsg: 0]
```

JavaScript syntax:

```
trace (myDB.createDB("E:\mydocuments\newdatabase"))  
  
// [#errorMsg: 601]  
  
trace ( myDB.explainError(601))  
  
// "Can not create database file"  
  
trace (myDB.createDB("C:\mydocuments\newencrypteddatabase", "mysecretkey"  
)  
)  
  
// [#errorMsg: 601]
```

`xtrainsance.changeKey(string newencryptionkey)` - Changes the encryption key for the current database file. This function should be used after the database file is opened with `openDB` or `createDB`. To remove encryption and make the file compatible with standard SQLite 3.x databases you can use an empty string as the new key. The encryption key should be a string longer than 6 standard characters (numbers or letters). Returns a Lingo property list, containing any errors generated by the command as the `#errorMsg` property.

Example:

```
put myDB.changeKey("mynewkey")

-- [#errorMsg: 0]

--Setting an empty key removes encryption

put myDB.changeKey("" )

-- [#errorMsg: 0]
```

JavaScript syntax:

```
trace (myDB.changeKey("mynewkey"))

// [#errorMsg: 0]

// Setting an empty key removes encryption

trace (myDB.changeKey("" ))

// [#errorMsg: 0]
```

`xtrainsance.executeSQL(string sqlquery , (optional) list values)` - This function is the heart of the Arca Database Xtra. It can be used to create tables and indexes, update, delete or insert data, and to retrieve information from the database, all using standard SQL queries. The `sqlquery` parameter is the `sqlquery` to be executed. An optional list of Lingo values may be passes as a second parameter, and the Xtra will automatically replace all occurrences of the interrogation mark (?) character in the SQL query with the corresponding values in the list, already quoted and prepared for SQL compliance. A list of SQL commands supported by Arca can be found at

## Online Help

the Supported SQL table, and is directly inherited from the strong SQL support offered by the SQLite engine used in Arca.

This command returns a property list containing the following properties:

**#errorMsg:** any error messages generated during the execution of the command. You can use the `explainError` function to retrieve a text explanation of the error, or consult the error table in this guide.

**#rowschanged:** the number of rows affected by the SQL command. This value is only useful when UPDATE, INSERT and DELETE commands are issued, and contains the number of records changed after the command was executed.

**#rows:** the records returned after execution of an SQL command, usually SELECT. Each record is returned as an individual list in the #rows list value, so the developer can use `rows.count` to determine the number of records available.

**#columns:** a list containing the name of the fields retrieved by your query, usually a SELECT statement.

**#columnstype:** a list with the types of each field retrieved, matching the number of elements in the #columns list.

For more information about detailed usage of the `executeSQL` function please read the [Inserting, Deleting and Updating](#) and [Retrieving Information](#) pages in this user guide.

Lingo examples:

```
put myDB.executeSQL("INSERT INTO users VALUES(?,?,?)", [member(1).text,
gMyAge, sprite(2).member.picture])
```

```
-- [#rowschanged: 1, #columns: [], #columnstype: [], #rows: [], #errorMsg: 0]
```

```
put gDB.executeSQL("SELECT name, age FROM users WHERE age<=18")
```

```
-- [#rowschanged: 0, #columns: ["name", "age"], #columnstype: ["text", "integer"],
#rows: [{"John", 15}, {"Mary", 13}, {"Paulo", 10}], #errorMsg: 0]
```

JavaScript syntax examples:

```
trace (myDB.executeSQL("INSERT INTO users VALUES(?,?,?)",
list(member(1).text, gMyAge, sprite(2).member.picture))
```

```
// [#rowschanged: 1, #columns: [], #columnstype: [], #rows: [], #errorMsg: 0]
```

## Online Help

```
trace (gDB.executeSQL("SELECT name, age FROM users WHERE age<=18"))
```

```
// [#rowschanged: 0, #columns: ["name", "age"], #columnntype: ["text", "integer"],  
#rows: [{"John", 15}, {"Mary", 13}, {"Paulo", 10}], #errorMsg: 0]
```

xtrainstance.createSelection(string sqlquery , (optional) list values) - This function has the same syntax as the executeSQL function. The sqlquery parameter is the SQL query to be executed. An optional list of Lingo values may be passes as a second parameter, and the Xtra will automatically replace all occurrences of the interrogation mark (?) character in the SQL query with the corresponding values in the list, already quoted and prepared for SQL compliance. This function is used with an SQL query containing a SELECT statement, in order to produce a selection to be explored with the getRows and getField function.

This command returns a property list containing the following properties:

**#errorMsg:** any error messages generated during the execution of the command. You can use the explainError function to retrieve a text explanation of the error, or consult the error table in this guide.

**#selectionid:** a numeric id of the selection generated by this SQL query, to be used with the getRows and getField functions to access records.

**#numrows:** the number of rows contained in the selection.

**#columns:** a list containing the name of the fields retrieved by your query, usually a SELECT statement.

**#columnntype:** a list with the types of each field retrieved, matching the number of elements in the #columns list.

For more information about detailed usage of the createSelection function please read the second part of the [Retrieving Information](#) page in this user guide.

Lingo:

```
mySelection = myDB.createselection("SELECT * FROM users")
```

```
put mySelection
```

```
-- [#errorMsg: 0, #selectionid: 27881928, #numrows: 1, #columns: ["name",  
"age"], #columnntype: ["text", "text"]]
```

## Online Help

JavaScript syntax :

```
mySelection = myDB.createselection("SELECT * FROM users")
```

```
trace (mySelection)
```

```
// [#errorMsg: 0, #selectionid: 27881928, #numrows: 1, #columns: ["name", "age"],  
#columnstype: ["text", "text"]]
```

`xtrinstance.getRows(integer selectionid, integer firstRow, integer lastRow)` - Used to retrieve specific rows from a previously created selection value. This can be used to control how many rows are retrieved and accessing only the data that needs to be displayed at any given time, optimizing the performance of your application.

This command returns a property list very similar to the one returned by the `executeSQL` command, containing the following properties:

**#errorMsg:** any error messages generated during the execution of the command. You can use the `explainError` function to retrieve a text explanation of the error, or consult the error table in this guide.

**#rows:** the records specified by the `firstRow` and `lastRow` values. Each record is returned as an individual list in the `#rows` list value, so the developer can use `rows.count` to determine the number of records available.

**#columns:** a list containing the name of the fields retrieved by your query.

**#columnstype:** a list with the types of each field retrieved, matching the number of elements in the `#columns` list.

For more information about detailed usage of this function please read the second part of the [Retrieving Information](#) page in this user guide.

Lingo:

```
mySelection = myDB.createselection("SELECT * FROM users")
```

```
put myDB.getRows(mySelection.selectionid , 1, 3)
```

```
-- [#columns: ["name", "age", "address"], #columnstype: ["text", "numeric", "text"],  
#rows: [{"john", 22, "178 Bond Street"}, {"mary", 70, "23 Circle Park"},  
{"anthony", 31, "4567 Privet Drive"}], #errorMsg: 0]
```

## Online Help

JavaScript syntax :

```
mySelection = myDB.createselection("SELECT * FROM users")

trace (myDB.getRows(mySelection.selectionid , 1, 3))

// [#columns: ["name", "age", "address"], #columnType: ["text", "numeric", "text"],
#rows: [{"john", 22, "178 Bond Street"}, {"mary", 70, "23 Circle Park"},
{"anthony", 31, "4567 Privet Drive"}], #errorMsg: 0]
```

xtrainstance.getField(integer selectionid, integer rownumber, string fieldname) -  
Used to retrieve a specific Lingo value from a selection. This command takes the  
row number and the name of the field to be retrieved, and returns a Lingo value  
directly. Field names are not case sensitive. This can be more convenient since the  
developer does not need to parse a result list.

This command returns the Lingo value requested if all parameters are valid, or  
<Void> if there is a problem (for example you specify a field name that does not  
exist, or an invalid selection id.)

For more information about detailed usage of this function please read the second  
part of the [Retrieving Information](#) page in this user guide.

Lingo:

```
mySelection = myDB.createselection("SELECT * FROM users")

put myDB.getField(mySelection.selectionid, 200, "age")

-- 32

put myDB.getField(mySelection.selectionid, 200, "nonvalidfieldname")

--<Void>
```

JavaScript syntax :

```
mySelection = myDB.createselection("SELECT * FROM users")

trace (myDB.getField(mySelection.selectionid, 200, "age"))

// 32

trace (myDB.getField(mySelection.selectionid, 200, "nonvalidfieldname"))

//<Void>
```

`xtrainsance.freeSelection(integer selectionid)` - This function frees memory allocated to the specified selection object. You should also release all selections allocated by `createSelection` commands when you no longer need them. The Xtra will automatically release memory when the Xtra instance object is destroyed or a database file is closed, but it is good practice to control memory usage at all times.

A developer can also use the parameter 0 as the `selectionid` to cause the Xtra to release memory and discard all selections currently active for this Arca instance.

This command returns a Lingo property list, containing any errors generated by the command as the `#errorMsg` property.

For more information about usage of this function please read the second part of the [Retrieving Information](#) page in this user guide.

Lingo:

```
mySelection = myDB.createselection("SELECT * FROM users")

put myDB.getField(mySelection.selectionid, 200, "age")

-- 32

put myDB.freeSelection(mySelection.selectionid)

--[errorMsg: 0]
```

JavaScript syntax :

```
mySelection = myDB.createselection("SELECT * FROM users")

trace (myDB.getField(mySelection.selectionid, 200, "age"))

// 32

trace (myDB.freeSelection(mySelection.selectionid))

//[errorMsg: 0]
```

`xtrainsance.closeDB()` - Close the database file associated with this instance of the Arca Database Xtra. Returns a Lingo property list, containing any errors generated by the command as the `#errorMsg` property. Database files are also automatically closed by Arca when the Xtra instance object is destroyed.

Lingo:

```
put myDB.closeDB()
```

```
-- [#errorMsg: 0]
```

JavaScript syntax :

```
trace (myDB.closeDB())
```

```
// [#errorMsg: 0]
```

`xtrinstance.compactDB()` - Compacts the database file on disk, removing any empty spaces that are left unused when records are deleted and no new records are created to fill the space. This is usually not necessary unless you are deleting a large number of records and do not plan on adding new data: empty space in the database file will be reused automatically when new data is inserted. Returns a Lingo property list, containing any errors generated by the command as the `#errorMsg` property. Compacting the database is a relatively expensive operation, and it takes some time to complete. It also can change the value of the internal ROWID field, so if you are depending on ROWID as a fixed key this operation should not be used (most applications with this requirement should use an explicit INTEGER PRIMARY KEY field, however).

Lingo:

```
put myDB.compactDB()
```

```
-- [#errorMsg: 0]
```

JavaScript syntax :

```
trace (myDB.compactDB())
```

```
//[#errorMsg: 0]
```

`xtrinstance.getDBSchema()` - This utility command is meant to be used for tools that need to inspect the structure of the database using Lingo. It returns a property list containing the `#rows` value: each row is one object in the database structure (a table or an index), and they type of object, name, referring table (for indexes) and sql command used to create it are displayed. Developers working with database files may prefer to use the DATABASE STRUCTURE view of the Arca Database



## Online Help

Browser application for easier inspection of Arca database structures.

Lingo:

```
put myDB.getDBschema()
```

```
-- [#errorMsg: 0, #rowschanged: 0, #columns: ["type", "name", "tbl_name", "sql"],  
#columnType: ["text", "text", "text", "text"], #rows: [{"table", "users", "users",  
"create table users(name text, age text)"}]]
```

JavaScript syntax :

```
trace (myDB.getDBschema())
```

```
// [#errorMsg: 0, #rowschanged: 0, #columns: ["type", "name", "tbl_name", "sql"],  
#columnType: ["text", "text", "text", "text"], #rows: [{"table", "users", "users",  
"create table users(name text, age text)"}]]
```

`xtrinstance.getLastInsertRowid()` - This utility command returns the rowid of the record created by the last INSERT SQL command, or <VOID> if there has been no inserts in the current database session. It is meant to be used when your application wants to insert an empty record in the database to be filled later, and you need some way to keep a reference to the newly created row.

Lingo:

```
myDB.executeSQL("INSERT INTO users VALUES('john',22,'2345678')")
```

```
put myDB.getlastinsertrowid()
```

```
-- 1345666
```

JavaScript syntax:

```
myDB.executeSQL("INSERT INTO users VALUES('john',22,'2345678')")
```

```
trace (myDB.getlastinsertrowid())
```

```
// 1345666
```

`xtrinstance.setEncoding(symbol side, symbol newencoding)` - In Director 8.5, MX and MX 2004, Arca by default uses the #MACROMAN client encoding when

## Online Help

translating data on the Macintosh platform, and the #LATIN1 client encoding when translating string data on Windows. The default database encoding is UTF8. In Director 11 (all platforms) the client encoding is also UTF8, so no translation is needed. For almost all applications there is no need to change the default encoding, so this function should never be used. However, if you are trying to use Arca to store double byte text on older versions of Director it might be necessary to disable the translation, so string data will stored exactly as passed. Valid values for the side parameter are #CLIENT and #DB. Valid values for the newencoding parameter are #UTF8, #LATIN1, #MACROMAN and #NONE. Please check the [additional features](#) page for more information on database encodings.

Lingo:

```
myDB.setEncoding(#CLIENT, #NONE)
```

```
myDB.setEncoding(#DB, #NONE)
```

JavaScript syntax:

```
//for compatibility with databases converted from Arca 1.x
```

```
myDB.setEncoding(symbol("DB"), symbol("LATIN1"))
```



## ARCA DATABASE XTRA HELP: ERROR CODES

The following is a list of error codes returned by the Arca Database Xtra. It is important to notice that negative error codes are not Arca errors, but Director standard error codes.

- 0 "OK"
- 1 "SQL error"
- 2 "Internal logic error"
- 3 "Access permission denied"
- 4 "Callback routine requested an abort"
- 5 "The database file is locked"
- 6 "A table in the database is locked"
- 7 "Memory allocation failed"
- 8 "Attempt to write a readonly database"
- 9 "Operation terminated by interrupt request"
- 10 "Some kind of disk I/O error occurred"
- 11 "The database disk image is malformed"
- 12 "Table or record not found"
- 13 "Insertion failed because database is full"
- 14 "Unable to open the database file"
- 15 "Database lock protocol error"
- 16 "Database table is empty"
- 17 "The database schema changed"
- 18 "Too much data for one row of a table"
- 19 "Abort due to constraint violation"

## Online Help

- 20 "Data type mismatch"
- 21 "Library used incorrectly"
- 22 "Uses OS features not supported on host"
- 23 "Authorization denied"
- 24 "Auxiliary database format error"
- 25 "Bind parameter out of range"
- 26 "Database file is encrypted or is not valid"
- 100 "Another row ready"
- 101 "Callback has finished executing"
- 600 "Can not open database file"
- 601 "Can not create database file"
- 602 "Database opened in read-only mode"
- 603 "The database is already opened"
- 604 "Can not find the database file"
- 605 "Output file already exists"
- 606 "Can not find input file"
- 608 "Invalid registration information"
- 609 "There is no database opened"
- 610 "Can not create output file"
- 611 "Can not open input file"
- 612 "Wrong number of arguments or items in the list"
- 613 "Function requires a list argument"
- 614 "Not a valid type for database insertion"
- 615 "Not a valid selection id"
- 616 "Not a valid field name"

617 "Invalid encoding symbol"

618 "Invalid encoding side symbol"



## ARCA DATABASE XTRA HELP: SQL SUPPORT

Arca incorporates a customized version of the SQLite database engine, adapted to work with Director native types, binary values and Lingo, and offering additional features as UTF8 translation, compression and encryption. However it still maintains almost all SQL functionality offered by the SQLite engine, which includes the majority of SQL92 features. SQLite support for standards is considered one of the best in the software industry, including advanced features like transactions, which are not available on any other database solution for Director at this time.

We recommend that you check the [SQL resources page](#) at our site periodically for updated links to SQL tutorials and articles, related to Arca, generic SQL and SQLite specific commands.

Below is a partial list containing the main SQL keywords and functions supported in Arca:

BEGIN TRANSACTION

COMMIT TRANSACTION

CREATE INDEX

CREATE TABLE

DELETE

DROP INDEX

DROP TABLE

END TRANSACTION

INSERT

ROLLBACK TRANSACTION

SELECT

UPDATE



## ARCA DATABASE XTRA HELP: CREATING PROJECTORS

Arca Database Xtra can be used to create projector in all operational systems and platforms supported by Director 8.5, Director MX, Director MX 2004 and Director 11.

### CREATING A CROSS-PLATFORM OR STANDARD PROJECTOR

(Director 11.5, 11 and MX 2004)

Director 11.5, 11 and MX2004 include the ability to create Windows projectors when hosted on Mac OS X, and vice-versa. The OSX version of Director MX 2004 can create projectors for older versions of MacOS (8 and 9) as well. However, this only works correctly if Director is configured to locate and include the proper Xtra files for "the other" platform. Please make sure the appropriate files are installed in the Configuration\Cross platform resources Director folder according to the installation notes, and also make sure the Configuration\xtrainfo.txt file contains information about Arca Database Xtra (this procedure is also covered in the installation instructions)

After the Xtra is installed, make sure the Xtra is included in your Director movie. With your .dir file opened, please select the MODIFY->MOVIE->XTRAS menu item. Click the ADD button , and include the Arca Database Xtra (Arca.x32 on Windows) in the first movie used in your projector. Save the file.

Assuming the cross-platform files are already installed and configured correctly you can now invoke the FILE->PUBLISH SETTINGS menu item to configure the parameters for your projector. You can then configure what types of projectors will be created. You should disable the option to create a Shockwave file, as Arca Database Xtra is not available for Shockwave.

In the FILES tab you can add additional Director movies to your projector. To finalize just click the PUBLISH button. Director saves the publishing settings with your Director movie, and future projectors can be created simply by choosing the FILE->PUBLISH menu item.

### CREATING A STANDARD WINDOWS OR MACOS PROJECTOR

## Online Help

(Director 8.5 and MX)

Director 8.5 and MX can only create native projectors. Windows projectors need to be created on a Windows machine, and Macintosh projectors need to be created on a Macintosh computer. However, Director movies (.dir files) containing Arca Database Xtra scripts don't need any modification in order to work on both platforms. A developer can work primarily on the Mac and only transfer the final .dir file to Director for Windows in order to create a Windows projector, or vice-versa. The only requirement is to install the Xtra on both platforms. Instructions for installing the Xtra can be found in the download packages. Please notice that Arca Database Xtra serial numbers are cross-platform: you can use the same serial number with the `arcaregister()` function on your movie to register the software on both Mac and Windows.

After the Xtra is installed, make sure the Xtra is included in your movie. With your .dir file opened, please select the **MODIFY->MOVIE->XTRAS** menu item. Click the **ADD** button, and include the Arca Database Xtra (Arca.x32 on Windows) in the first movie used in your projector. Save the file. Now you just need to select **CREATE PROJECTOR** from the **FILE** menu in Director to create your projector, and Arca will be included in it.

**TIP:** You can also deliver the Xtra file in a folder named **XTRAS**, located in the same directory of your Projector, if you do not want to embed the Xtra file into your projector. This is recommended for faster startup of your program.





## ARCA DATABASE XTRA HELP: HOW TO ORDER & REGISTER

The unregistered version of Arca Database Xtra is fully-functional and may be used for evaluation, nonprofit and educational purposes only: commercial distribution is strictly prohibited. A registered version of the Xtra can be used in commercial products, and may be purchased online at [xtras.tabuleiro.com](http://xtras.tabuleiro.com), using a secure server. At our web site you can also consult our purchase policy, purchase instructions, payment, delivery and security methods.

If you decide to buy the Xtra you don't need to download a new copy of the software. After your order is processed you will receive an e-mail with a serial number to register the software you've already installed on your machine.

To register the Xtra you should use the `arcaregister()` function, usually called at the startup of your movie, or before an Arca database is created or opened. More information about specific syntax can be found at the [Xtra Functions](#) page. Please keep your serial number archived for future reference.



ARCA DATABASE XTRA HELP: LICENSING & AVAILABILITY

Arca Database Xtra is a commercial product. Current price and updated information can be found at [xtras.tabuleiro.com](http://xtras.tabuleiro.com). If your product provides printed documentation and package we ask you to kindly include the following copyright information:

Arca Database Xtra(tm) (c) Tabuleiro Prod. Ltda 2003-2008

All Rights Reserved

No royalty-fees are required for a distribution of the Xtra with your product.



ARCA DATABASE XTRA HELP: TECHNICAL SUPPORT

Please use the Your Account section available at our web site [xtras.tabuleiro.com](http://xtras.tabuleiro.com) to submit your questions. The site also contains Technotes and other resources that can help you identify and solve the most common problems quickly.