

L3 TECHNOLOGIES

dBASE File Driver

Copyright © 1998, L3 Technologies, Ltd. All rights reserved.
<http://www.l3.u-net.com>

CONTENTS

| | |
|---|-----------|
| INTRODUCTION | 4 |
| LICENSE AGREEMENT | 5 |
| INSTALLATION & CONFIGURATION | 8 |
| PURCHASING AND REGISTERING THE DBASE FILE DRIVER | 8 |
| LICENSING THE BORLAND DATABASE ENGINE | 8 |
| GENERAL CHARACTERISTICS | 10 |
| FILE SPECIFICATIONS/MAXIMUMS | 10 |
| SUPPORTED DATA TYPES | 11 |
| DBASE DATA TYPES..... | 11 |
| BLOB TYPES | 11 |
| SUPPORTED KEY AND INDEX DEFINITIONS | 12 |
| SPECIFYING DBASE INDEXES | 12 |
| DEFAULT INDEX CREATION RULES USED BY THE DRIVER | 12 |
| DBASE FUNCTIONS SUPPORTED IN KEY AND INDEX EXPRESSIONS | 13 |
| DRIVER STRINGS AND SEND COMMANDS | 16 |
| ABOUT | 16 |
| ALIAS | 16 |
| BDETRACE..... | 16 |
| CBNUMFIX..... | 16 |
| CLIENTNAME..... | 16 |
| DBHANDLE | 17 |
| DBOPENMODE..... | 17 |
| DBSHAREMODE..... | 17 |
| DBTYPE | 17 |
| DRIVER | 17 |
| INIFILE | 18 |
| INDEXBLOCKSIZE..... | 18 |
| LANGUAGE | 18 |
| LEVEL | 18 |
| LOCALINIT..... | 18 |
| MEMOBLOCKSIZE..... | 18 |
| POSITIONSIZE | 18 |
| PRIVATEDIR | 19 |
| QBE | 19 |
| SEQCURSOR | 19 |
| SQL | 19 |
| STRICT | 19 |
| TABLECURSOR | 19 |
| VERENGINE | 20 |
| VERINTERFACE | 20 |
| WORKDIR..... | 20 |
| SQL SUPPORT | 21 |
| CURSORS | 21 |

| | |
|--|-----------|
| ERROR HANDLING | 23 |
| NULL FIELD HANDLING | 24 |
| FOXPRO SUPPORT | 25 |
| TRANSACTION SUPPORT | 26 |
| UTILITIES AND EXAMPLES | 27 |
| DBINFO | 27 |
| DBF2TXD | 27 |
| TECHNICAL SUPPORT & RESOURCES | 28 |
| UNSUPPORTED OR MODIFIED FUNCTIONS & ATTRIBUTES..... | 29 |
| LIMITATIONS | 30 |
| ACKNOWLEDGEMENTS..... | 31 |

Introduction

The dBASE File Driver is one of the most sophisticated Clarion File Drivers yet developed.

The dBASE File Driver provides full native access to dBASE tables from Clarion for Windows applications. dBASE table access is achieved using the Borland Database Engine (BDE). This is the same engine that is used in the dBASE Database, so full compatibility is ensured.

A copy of the BDE Redistributables is included. If you develop and distribute software using the dBASE File Driver, you must redistribute this same copy of the BDE subject to certain restrictions (see Licensing the Borland Database Engine).

The dBASE File Driver is written using the Clarion File Driver Kit, ensuring complete compatibility with Clarion for Windows. In addition the Professional variant of the driver incorporates new technology that enables applications to dynamically create, open and access dBASE tables without defining the file structure at compile time. This enables developers to produce general purpose applications that can operate on any dBASE table.

The Professional variant of the dBASE Driver offers full support for Local SQL through the use of the Prop:SQL file property. Support for Query By Example (QBE) is provided by a Prop:QBE property. In addition a special form of file declaration, is supported which allows the efficient use of SQL style cursors in applications.

An extended error handling mechanism is incorporated, which provides access to the BDE error information as well as posting the appropriate Clarion error code. In addition there are a number of driver specific errors that are used to pinpoint specific error conditions during application development.

The dBASE Driver offers an unparalleled range of features and functionality which ease the development of applications previously impossible with conventional file drivers.

License Agreement

This license agreement is a legal agreement between you (either an individual or a single entity) and L3 Technologies Ltd. for the software product identified above, which includes computer software and may include associated media, printed materials, and "online" or electronic documentation ("SOFTWARE PRODUCT"). By installing, copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this license agreement. If you do not agree to the terms of this license agreement, do not install or use the SOFTWARE PRODUCT; you may, however, return it to your place of purchase for a refund.

SOFTWARE PRODUCT LICENSE

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

L3 Technologies Ltd. grants the following rights to the licensed, registered user of the SOFTWARE product, subject to all of the conditions in this license agreement:

You may install one copy of this SOFTWARE PRODUCT for use. You may make archive copies of the original distribution files for the sole purpose of backing up the SOFTWARE PRODUCT and safeguarding your investment. The SOFTWARE PRODUCT may be used by any number of people, and may be freely moved from one computer or location to another so long as there is no possibility of it being used by more than one person at a time. The SOFTWARE PRODUCT is licensed as a single product, except as provided below, its component parts may not be separated for use by more than one person at a time.

The SOFTWARE PRODUCT may include files intended for distribution by you to users of the programs you create. These files are specifically designated as "Redistributable" in the "readme" file for the SOFTWARE PRODUCT ("REDISTRIBUTABLES"). You are authorized to reproduce and distribute exact copies of the REDISTRIBUTABLES for the sole purpose of executing application programs that you have developed using the SOFTWARE PRODUCT. Under no circumstances may you distribute any copies of the REDISTRIBUTABLES separately.

You may not alter or remove any copyright or trademark notices or identifying screens contained in any of the libraries, source code or REDISTRIBUTABLES comprising the SOFTWARE PRODUCT.

Regardless of any modifications that you make, you may not distribute any files (particularly source code and other non-executable files) except REDISTRIBUTABLES.

You will remain solely responsible to anyone receiving your programs for support, service, upgrades, technical or other assistance, and such recipients will have no right to contact L3 Technologies for such services or assistance.

You will indemnify and hold L3 Technologies, and its suppliers harmless from and against any claims or liabilities arising out of the use, reproduction, or distribution of your programs.

If you have purchased an upgrade version of the SOFTWARE PRODUCT, it constitutes a single product with the SOFTWARE PRODUCT that you upgraded.

The upgrade and the SOFTWARE PRODUCT cannot both be available for use by two different people at the same time, and cannot be transferred separately.

TITLE

All L3 Technologies intellectual property rights, libraries, source code, REDISTRIBUTABLES and other files comprising the SOFTWARE PRODUCT remain the exclusive property of L3 Technologies Ltd. or its suppliers.

TRANSFER OF LICENSE

You may permanently transfer all of your rights under this license agreement, provided the recipient agrees to the terms of this license agreement, and you transfer all software, media and documentation comprising the SOFTWARE PRODUCT, and you transfer or destroy all copies in any form.

TERMINATION OF LICENSE

Without prejudice to any other rights, L3 Technologies may terminate this license agreement if you fail to comply with the terms and conditions of this license agreement. In such event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts.

LIMITED WARRANTY

L3 Technologies Ltd. warrants the physical media and physical documentation provided by L3 Technologies Ltd. to be free from defects in materials and workmanship for a period not exceeding 30 days from the original purchase date. If L3 Technologies Ltd. receives notification within the warranty period of defects in materials or workmanship, and determines that such notification is correct, L3 Technologies Ltd. will replace the defective media or documentation.

DISCLAIMER OF WARRANTY

L3 Technologies Ltd. expressly disclaims any warranty for the SOFTWARE PRODUCT. The SOFTWARE PRODUCT and any related documentation is provided "as is" without warranty, representation or condition of any kind, either express or implied, including, without limitation, any implied warranty or condition of merchantability, fitness for a particular purpose, or noninfringement.

Specifically, L3 Technologies Ltd. makes no representation or warranty that the SOFTWARE PRODUCT, or any components of the SOFTWARE PRODUCT, including any related documentation are "error free" or meet any user's particular standards requirements or needs. In all events, any implied warranty, representation, condition or other term is limited to the physical media or documentation and is limited to the 30-day duration of the limited warranty.

The entire risk arising out of use or performance of the SOFTWARE PRODUCT remains with you.

In no event shall L3 Technologies Ltd. or its suppliers be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information or data, or any other pecuniary loss) arising out of the use of or inability to use this SOFTWARE PRODUCT, even if L3 Technologies Ltd. has been advised of the possibility of such damages.

L3 Technologies Ltd. is not responsible for, and does not make any representation, warranty, or condition concerning product, media, software, or documentation not manufactured by L3 Technologies Ltd. such as third parties' programs which are developed using L3 Technologies software or which include L3 Technologies programs, library code or files.

HIGH RISK ACTIVITIES

The SOFTWARE PRODUCT is not fault-tolerant and is not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as (but not limited to) the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Software could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). L3 Technologies Ltd. and its suppliers specifically disclaim any express or implied warranty of fitness for High Risk Activities.

MISCELLANEOUS

This license agreement is governed by the laws of the United Kingdom.

Installation & Configuration

The dBASE File Driver SETUP program will install the driver files, utilities, examples and documentation into your Clarion directory structure. A program group will be created to provide easy access to the documentation and utilities.

Before the dBASE File Driver can be accessed from the Clarion IDE, it is necessary to install it in the Database Driver Registry. To do this select the **Database Driver Registry...** item on the **Setup** menu, click on the **Add** button and choose the appropriate 16-bit dBASE File Driver DLL for the version of Clarion you are running:

| | |
|--------------------------------|---|
| Clarion for Windows 2.0 | CW2PDBF16.DLL (Professional) CW2DBS16.DLL (Standard) |
| Clarion 4.0 | C4DBF.DLL (Professional) C4DBS.DLL (Standard) |
| Clarion 5.0 | C5DBF.DLL (Professional) C5DBS.DLL (Standard) |

It may be necessary to close the IDE and restart in order for the changes to take effect.

A number of environmental options may be configured using the configuration utility installed with the Borland Database Engine (BDE). The utility may be named BDECFG.EXE, BDECFG32.EXE or BDEADMIN.EXE depending on the version of the BDE you have installed. Consult the on-line help for this utility for additional information.

Purchasing and Registering The dBASE File Driver

The dBASE File Driver is delivered in two stages. Firstly you should download and install the demo versions of the L3 File Drivers. This will install the documentation, examples, utilities and the BDE redistributables. You will also have the opportunity to evaluate the driver and ensure that it suites your requirements. You may then purchase the driver on-line or by Electronic Software Delivery (ESD). Note that only the File Driver binaries themselves are included in the ESD delivery to reduce overheads, so installing the demonstration version is a pre-requisite.

You may download and purchase the software from one of the dealers listed on the L3 Technologies web site (<http://www.l3.u-net.com>).

By registering the dBASE File Driver you will be able to personalise the details that appear on the About screen for the driver which may be useful for distribution purposes.

You will receive your registration details shortly after you have made your purchase, otherwise contact your supplier. You must then use the L3REG.EXE application supplied with the driver to modify the driver. Simply load the application from Explorer or the File Manager and follow the on-screen instructions.

Licensing the Borland Database Engine

Any applications developed using the dBASE File Driver must include the Borland Database Engine (BDE) in the distribution.

The Borland Database Engine (BDE) supplied with the dBASE Driver must be shipped together with appropriate BDE Diagnostic utility (DBINFO16.EXE or DBINFO32.EXE) also supplied. This is to satisfy the Borland License Agreement, which states that the BDE may only be shipped with applications that are developed with a Borland development tool. The BDE Diagnostic utility is developed with Borland C++.

Full details of precisely which files need to be redistributed with your applications are included in the accompanying README.TXT file.

General Characteristics

The dBASE File Driver supports dBASE III, III+, IV, dBASE for Windows and FoxPro 2.5. The default data file extension is .DB.

Keys and indexes exist separately from the main data file, and have a .NDX or .MDX extension.

dBASE supports only fixed length records. Memos are not supported, however BLOBs are. BLOBs are stored in a separate file with a .DBT extension.

File Specifications/Maximums

| | |
|------------------------|--|
| File Size: | 2 billion bytes |
| Records per File: | 1 billion |
| Record Size: | 4000-32767 depending on type and version |
| Field Size: | 254 |
| Fields per Record: | 254-1024 depending on type and version |
| Keys/Indexes per File: | 1 MDX with up to 47 tags + 10 NDX's |
| Key Size: | 254 |
| Memo fields per File: | 0 |
| Memo Field Size: | 0 |
| BLOB fields per File: | 255 |
| BLOB Size: | BLOBs limited only by available memory |
| Open Data Files: | Operating system dependent |

Supported Data Types

Some Clarion data types represent more than one DBASE type, for this reason it is necessary to augment the field definition to specify the DBASE type. Any field type may be augmented, though it is only necessary to augment fields with more than one Clarion representation and Blobs.

Augmented field definitions take the form:

```
FieldLabel ClarionType,NAME('FieldLabel->DBASEType')
```

Where ClarionType is an appropriate supported Clarion data type, and DBASEType is the letter normally used within DBASE to designate a given type. The following tables detail the Clarion data type equivalencies for DBASE types.

DBASE Data Types

| | | | |
|-----------------|---|-------------------------------------|------------|
| Character | C | STRING(n) | |
| Float | F | REAL, NAME('Field->F(n,p)') | |
| | | DECIMAL, NAME('Field->F(n,p)') | |
| Number | N | DECIMAL(n,p), NAME('Field->N(n,p)') | |
| | | DECIMAL(n,p) | |
| | | REAL, NAME('Field->N(n,p)') | |
| Date | D | DATE | |
| Logical | L | BYTE | |
| Double † | O | REAL | |
| Long † | I | LONG | |
| AutoIncrement † | + | LONG, NAME('Field->+') | |
| DateTime † | @ | STRING(8), NAME('Field->@') | |
| | | GROUP(TBDETimeStamp). | |
| LockInfo | K | STRING(n), NAME('Field->K') | ! n = 8-24 |

BLOB Types

| | | | |
|--------|---|-----------------------------|--|
| Memo | M | ULONG, NAME('Blob->M') | |
| | | STRING(10), NAME('Blob->M') | |
| Binary | B | ULONG, NAME('Blob->Bn') | |
| | | STRING(10), NAME('Blob->B') | |
| OLE | G | ULONG, NAME('Blob->G') | |
| | | STRING(10), NAME('Blob->G') | |

† dBASE 7 type, requires 32-bit BDE version 3 and above

Supported Key and Index Definitions

dBASE indexes are all based on expressions using fields in the record. In the simplest case, an index is an expression containing just the name of the field on which the index is built. More sophisticated indexing requirements are satisfied using complex expressions involving functions and fields in the record. In dBASE IV and above, and FoxPro, indexes can include a subsetting expression which serves to filter the records in the index.

The dBASE File Driver supports all index types. Standard dBASE functions may be used in index expressions, but currently there is no support for user-defined functions or program variables.

Specifying dBASE Indexes

Where a key or index contains only a single field, or a simple concatenation of fields in the record structure, a conventional Clarion KEY or INDEX declaration will suffice. Complex index declarations are handled by placing the expression in the NAME() attribute for the KEY or INDEX. The syntax is as follows:

```
KeyLabel    KEY(FldList), NAME(IndexSpecifier)
```

Where FldList is the list of fields used in the index and IndexSpecifier has the following form in the case of non-compound indexes:

```
NdxPath->T[Expression]
```

And the following form in the case of compound indexes (dBASE IV and above and FoxPro):

```
TagName(BlockSize)->T[Expression],FOR[Expression]
```

T is the type of the expression and takes one of the following values:

- C Key expression returns a character value
- N Key expression returns a numeric value
- B Key expression returns a BCD value

Expression is an expression using fields in the record and possibly certain dBASE functions.

BlockSize is an optional value which is used to determine the size of the data blocks in a compound index at the point of creation.

The FOR[Expression] is an optional subsetting filter valid only for dBASE IV and above and FoxPro compound indexes.

Field names in expressions are the names of the fields as dBASE sees them. That is, they must be the external name of the field as specified in the NAME() attribute if it is used, otherwise it is the label of the field with no prefix.

Default Index Creation Rules Used by the Driver

In the simplest cases where the type of the index isn't specified using the NAME() attribute, the driver must make a decision about the type of dBASE Index to create. For dBASE IV and above there are two types of index file: a single index

file type which has the extension .NDX and a multiple index file which has the extension .MDX.

Single index files are not maintained, that is they must be updated using the BUILD() command, they sort in ascending order only and cannot contain subsetting or filter expressions. The name of the file is determined from the external name attribute or the label.

Multiple index files may contain several indexes. Each index is identified by a unique tag. The name of the tag is determined from the external name attribute or the label. The name of the index file is the same as the name of the table. Indexes are maintained, may sort in ascending or descending order and use subsetting expressions.

dBASE Functions Supported in Key and Index Expressions

The dBASE File Driver supports most dBASE functions and operators in index expressions. Full documentation for these functions is beyond the scope of this document, however a partial list together with a brief description is presented for reference.

| | |
|--------------------------|---|
| [...] | <i>indicates an optional parameter</i> |
| n | <i>indicates a numeric expression</i> |
| c | <i>indicates a character expression</i> |
| e | <i>indicates an arbitrary expression</i> |
| d | <i>indicates a date expression</i> |
| | |
| ABS(n) | <i>Returns the absolute value of a number.</i> |
| ACOS(n) | <i>Returns the radian value of the angle with cosine n.</i> |
| ANSI(c) | <i>Returns the ANSI equivalent of the specified OEM character sequence.</i> |
| ASC(c) | <i>Returns the numeric ASCII value of the specified character.</i> |
| ASIN(n) | <i>Returns the radian value of the angle whose sine is n.</i> |
| AT(c1,c2[,n]) | <i>Returns the n'th occurrence of the string c1 in c2. If unspecified, n defaults to 1.</i> |
| ATAN(n) | <i>Returns the radian value of the angle with tangent n.</i> |
| ATN2(n1,n2) | <i>Returns the radian value of the angle whose sine is n1 and whose cosine is n2.</i> |
| BITAND(n1,n2) | <i>Returns the value of the bitwise operation n1 AND n2.</i> |
| BITLSHIFT(n1,n2) | <i>Returns the value of n1 shifted left by n2 bits.</i> |
| BITOR(n1,n2) | <i>Returns the value of the bitwise operation n1 OR n2.</i> |
| BITRSHIFT(n1,n2) | <i>Returns the value of n1 shifted right by n2 bits.</i> |
| BITSET(n1,n2) | <i>Returns .T. if the n2th bit of n1 is set.</i> |
| BITXOR(n1,n2) | <i>Returns the value of the bitwise operation n1 XOR n2.</i> |
| CEILING(n) | <i>Returns the smallest integer that is equal to or greater than n.</i> |
| CHR(n) | <i>Returns the character equivalent of the specified ASCII value.</i> |
| COS(n) | <i>Returns the cosine of the angle n specified in radians.</i> |
| CTOD(c) | <i>Converts the specified character expression c to a date expression.</i> |
| DATABASE() | <i>Returns the name of the current default database.</i> |
| DATE() | <i>Returns the current system date.</i> |
| DAY(d) | <i>Returns the day of the month for the specified date.</i> |
| DIFFERENCE(c1,c2) | <i>Returns a number between 0 and 4 representing the phonetic difference between the strings c1 and c2. A</i> |

| | |
|-----------------------|--|
| | value of 4 indicated the least difference and a value of 0 indicates the greatest difference. |
| DOW(d) | Returns the day of the week for the specified date. |
| DTOC(d) | Converts the specified date expression <i>d</i> to a character expression. |
| DTOR(n) | Radian value of the angle <i>n</i> whose value is given in degrees. |
| DTOS(d) | Returns the specified date expression <i>d</i> as a character expression in the format YYYYMMDD. |
| ELAPSED(c2,c1) | Returns the number of seconds between the times <i>c1</i> and <i>c2</i> specified in HH:MM:SS format where <i>c2</i> is later than <i>c1</i> . |
| EMPTY(e) | Returns .T. if the expression or field is empty or blank. |
| EXP(n) | Returns <i>e</i> raised to the power <i>n</i> . |
| FIELD(n) | Returns the name of the field at the specified position in the record. |
| FLOOR(n) | Returns the largest integer less than or equal to <i>n</i> . |
| HTOI(c) | Returns the decimal equivalent of the specified hexadecimal value <i>c</i> . |
| ID() | Returns the name of the current user. |
| IIF(e1,e2,e3) | Returns <i>e2</i> if <i>e1</i> evaluates to .T. otherwise returns <i>e3</i> . |
| ISALPHA(c) | Returns .T. if the first character of <i>c</i> is alphabetic. |
| ISBLANK(e) | Returns .T. if the specified expression or field is blank. |
| ISLOWER(c) | Returns .T. if the first character of <i>c</i> is lowercase. |
| ISUPPER(c) | Returns .T. if the first character of <i>c</i> is uppercase. |
| ITOH(n) | Returns the hexadecimal value of the specified number. |
| LEFT(c,n) | Returns the first <i>n</i> characters from the string <i>c</i> . |
| LEN(c) | Returns the number of characters in the specified string. |
| LENNUM(n) | Returns the display length of the specified number. |
| LIKE(c1, c2) | Returns .T. if <i>c1</i> matches <i>c2</i> . The character expression <i>c1</i> may contain the wild-cards ? and *. |
| LOG(n) | Returns the logarithm base <i>e</i> of the specified number. |
| LOG10(n) | Returns the logarithm base 10 of the specified number. |
| LOWER(c) | Returns the string <i>c</i> converted to all lowercase. |
| LTRIM(c) | Returns the specified string or field with any leading spaces removed. |
| MAX(e1,e2) | Returns the maximum value of the expressions <i>e1</i> and <i>e2</i> . |
| MIN(e1,e2) | Returns the minimum value of the expressions <i>e1</i> and <i>e2</i> . |
| MOD(n1,n2) | Returns the remainder of <i>n1</i> divided by <i>n2</i> . |
| MONTH(d) | Returns the number of the month for the specified date. |
| OEM(c) | Returns the OEM equivalent of the specified ANSI character sequence. |
| PI() | Returns the approximate value of pi. |
| PROPER(c) | Returns the string <i>c</i> in proper-noun format. |
| RAT(c1,c2[,n]) | Returns the position of the <i>n</i> th occurrence of the string <i>c1</i> in <i>c2</i> . If unspecified, <i>n</i> defaults to 1. |
| REPLICATE(c,n) | Returns the string <i>c</i> repeated <i>n</i> times. |
| RIGHT(c,n) | Returns <i>n</i> characters from the end of the string <i>c</i> . |
| ROUND(n1,n2) | Returns the value of <i>n1</i> rounded to <i>n2</i> decimal places. |
| RTOD(n) | Returns the radian value of the angle <i>n</i> given in degrees. |
| RTRIM(c) | Returns the string <i>c</i> with trailing spaces removed. |
| SECONDS() | Returns a number of the form ss.hh where <i>ss</i> is the |

| | |
|-----------------------------|---|
| | <i>number of seconds, and hh is hundredths of a second, since midnight</i> |
| SIGN(n) | <i>Returns -1, 0 or 1 indicating that n is negative, 0 or positive.</i> |
| SIN(n) | <i>Returns the sine of the angle n specified in radians.</i> |
| SOUNDEX(c) | <i>Returns the 4-character soundex value of the specified string c.</i> |
| SPACE(n) | <i>Returns n space characters.</i> |
| SQRT(n) | <i>Returns the square root of n.</i> |
| STR(n1[,n2[n3[,c]]]) | <i>Returns the character string equivalent of n1 in a field of width n2 (defaults to 10) with n3 decimal places (defaults to 0) padding the start with c (defaults to a space).</i> |
| STUFF(c1,n1,n2,c2) | <i>Replaces the characters between position n1 and n2 in c1 with the string c2.</i> |
| SUBSTR(c,n1 [,n2]) | <i>Returns a substring of c starting at position n1 and ending at position n2 (defaults to the length of c).</i> |
| TAN(n) | <i>Returns the tangent of the angle n specified in radians.</i> |
| TIME([e]) | <i>Returns the system time in HH:MM:SS.hh format if e is specified, otherwise in HH:MM:SS format.</i> |
| TRIM(c) | <i>Returns the string c with trailing spaces removed.</i> |
| UPPER(c) | <i>Returns the string c converted to all lowercase.</i> |
| VAL(c) | <i>Converts the character expression c to a numeric expression.</i> |
| VERSION() | <i>Returns the currently running dBASE version as a character expression.</i> |
| YEAR(d) | <i>Returns the 4-digit year of the specified date.</i> |

Driver Strings and SEND commands

All send commands listed below may be used as driver strings (the second parameter in the DRIVER attribute on a FILE declaration) with the exception of 'ABOUT', 'DBHANDLE', 'POSITIONSIZE', 'SEDCURSOR' and 'TABLECURSOR'. Driver strings may be specified in one of two ways, comma separated:

```
f FILE, DRIVER('DBASE', 'ALIAS=MyTables, CLIENTNAME=PSW')
...
```

Preceded by '/':

```
f FILE, DRIVER('DBASE', '/ALIAS=MyTables /CLIENTNAME=PSW')
...
```

ABOUT

```
SEND(f, 'ABOUT')
```

The ABOUT command displays the drivers' 'About' screen. This lists the name and versions of the driver, and the details of the registered developer supplied when unlocking the driver. If the contact details are prefixed with mailto: (for an email address) or http: (for a web site address) then this will present an active link which, when clicked, will load the appropriate application using the ShellExecute() API. The L3 button acts as an active link to the L3 Technologies web site.

ALIAS

```
s" = SEND(f, 'ALIAS=dbalias')          f{PROP:DbAlias} = 'dbalias'
s" = SEND(f, 'ALIAS')                  s" = f{PROP:DbAlias}
```

The ALIAS command or property is used to reference the dBASE database alias. The alias must be specified before the table is opened. The dBASE database alias is used to group sets of related tables into a specific location. The alias is a configuration setting that is made using the IDAPICFG.EXE program.

BDETRACE

```
s" = SEND(f, 'BDETRACE=error.log')    f{PROP:BDETrace} = 'error.log'
```

The BDETRACE command or property is used to specify a trace file into which BDE trace information is written. This command has no effect unless the debugging version of the IDAPI interface is enabled. Consult the IDAPI help for advice on installing the debugging interface.

CBNUMFIX

```
SEND(f, 'CBNUMFIX')
```

There is a release of the CodeBase library that stores numeric fields as left-justified strings. dBASE stores numeric fields right-justified. Although dBASE is capable of reading left-justified numeric fields, the BDE by default expects numeric fields to be right-justified. The CBNUMFIX flag overrides this default behaviour allowing the driver to correctly read left-justified numeric fields. Note when a record is inserted or updated by the driver, all numeric fields are right-justified to maintain consistency with dBASE.

CLIENTNAME

```
s" = SEND(f, 'CLIENTNAME=client')    f{PROP:ClientName} = 'client'
```

```
s" = SEND(f, 'CLIENTNAME')          s" = f{PROP:ClientName}
```

The CLIENTNAME command or property is used to reference the client name for the current session. The client name must be specified before the table is opened. The client name is used for documentary purposes only (e.g. identifying the session that has locked a file).

DBHANDLE

```
h# = SEND(f, 'DBHANDLE')            h# = f{PROP:DbHandle}
```

The DBHANDLE command or property is used to obtain the table's database handle which may then be used when calling the IDAPI functions directly from Clarion. Care must be taken when calling the IDAPI directly, since this can affect file processing by the driver.

DBOPENMODE

```
s" = SEND(f, 'DBOPENMODE=0')        f{PROP:DbOpenMode} = 0  
i# = SEND(f, 'DBOPENMODE')          i# = f{PROP:DbOpenMode}
```

The DBOPENMODE command or property is used to reference the database open mode for the current session. The mode must be specified before the table is opened. The open mode for the database determines the access rights of any FILE opened on the database. If the database is opened in read-only mode, FILES will have only read-only access. The open mode values are as follows:

| | |
|---|----------------------|
| 0 | Read/Write (default) |
| 1 | Read-only |

DBSHAREMODE

```
s" = SEND(f, 'DBSHAREMODE=0')       f{PROP:DbShareMode} = 0  
i# = SEND(f, 'DBSHAREMODE')         i# = f{PROP:DbShareMode}
```

The DBSHAREMODE command or property is used to reference the database share mode for the current session. The mode must be specified before the table is opened. The share mode for the database determines the access rights of any FILE opened on the database. The share mode values are as follows:

| | |
|---|------------------|
| 0 | Shared (default) |
| 1 | Exclusive |

DBTYPE

```
s" = SEND(f, 'INDEXBLOCKSIZE=1024') f{PROP:IndexBlockSize} = 1024  
i# = SEND(f, 'INDEXBLOCKSIZE')      i# = f{PROP:IndexBlockSize}
```

The INDEXBLOCKSIZE command or property is used to determine the size of the blocks used to store data on the disk. The size is in the range 1024 to 16,384 for dBASE IV, 32,768 for dBASE for Windows.

DRIVER

```
s" = SEND(f, 'DRIVER=dBASE')         f{PROP:Driver} = 'dBASE'  
s" = SEND(f, 'DRIVER')              s" = f{PROP:Driver}
```

The DRIVER command is used to specify the driver that will be used to handle a specific table. It takes one of two values: 'dBASE', the default or 'FOXPRO'. If the BDE version 4.5 or above is in use, specifying 'FOXPRO' will load the BDE standard FoxPro driver, otherwise the dBASE driver will be used. The dBASE

driver can access FoxPro v2.5 and 2.6 tables. The driver must be specified before the table is opened.

INIFILE

```
s" = SEND(f, 'INIFILE=IDAPI.CFG')    f{PROP:IniFile} = 'IDAPI.CFG'  
s" = SEND(f, 'INIFILE')             s" = f{PROP:IniFile}
```

The INIFILE command or property is used to determine the configuration used by the Borland Database Engine. The configuration file must be specified before the table is opened.

INDEXBLOCKSIZE

```
s" = SEND(f, 'INDEXBLOCKSIZE=1024') f{PROP:IndexBlockSize} = 1024  
i# = SEND(f, 'INDEXBLOCKSIZE')      i# = f{PROP:IndexBlockSize}
```

The INDEXBLOCKSIZE command or property is used to determine the size of the blocks used to store index data on the disk. The size is in the range 1024 to 16,384 for dBASE IV, 32,768 for dBASE for Windows.

LANGUAGE

```
s" = SEND(f, 'LANGUAGE= DBWINUS0')  f{PROP:DbLanguage} = 'DBWINUS0'  
s" = SEND(f, 'LANGUAGE')            s" = f{PROP:DbLanguage}
```

The LANGUAGE command or property is used to load the specified BDE language driver.

LEVEL

```
s" = SEND(f, 'LEVEL=4')              f{PROP:Level} = 4  
i# = SEND(f, 'LEVEL')                i# = f{PROP:Level}
```

The LEVEL command or property is used to enforce support for a particular version of dBASE, overriding the BDE default setting. Valid values are in the range 3-7, depending on the version of the BDE in use. A value of 25 is used when accessing FoxPro tables.

LOCALINIT

```
s" = SEND(f, 'LOCALINIT=1')          f{PROP:LocalInit} = TRUE  
i# = SEND(f, 'LOCALINIT')            i# = f{PROP:LocalInit}
```

The LOCALINIT command or property is used to force local initialisation, if a value of TRUE (1) is specified, the default is FALSE (0).

MEMOBLOCKSIZE

```
s" = SEND(f, 'MEMOBLOCKSIZE=1024')  f{PROP:MemoBlockSize} = 1024  
i# = SEND(f, 'MEMOBLOCKSIZE')        i# = f{PROP:MemoBlockSize}
```

The MEMOBLOCKSIZE command or property is used to determine the size of the blocks used to store memo data on the disk. The default is 512 bytes.

POSITIONSIZE

```
i# = SEND(f, 'POSITIONSIZE')         i# = f{PROP:PositionSize}
```

The POSITIONSIZE command or property is used to determine the size of the position string required for the current sequence.

PRIVATEDIR

```
s" = SEND(f, 'PRIVATEDIR=C:\PRIV') f{PROP:PrivateDir} = 'C:\PRIVATE'  
s" = SEND(f, 'PRIVATEDIR') s" = f{PROP:PrivateDir}
```

The PRIVATEDIR command or property is used to set or get the private directory for the current session. The private directory must be specified before the file is opened.

QBE

```
s" = SEND(f, 'QBE=???') f{PROP:QBE} = '???'  
s" = SEND(f, 'QBE') s" = f{PROP:QBE}
```

The QBE command or property is used to submit a query or determine whether a previously submitted query used QBE. If a previously committed query was submitted using QBE the result is TRUE (1), otherwise it is FALSE (0). *This command is not available in the Standard variant of the driver.*

SEQCURSOR

```
s" = SEND(f, 'SEQCURSOR') s" = f{PROP:SeqCursor}
```

The SEQCURSOR command or property is used to obtain the table's sequential access handle. The sequential access handle is determined by any SET() call and is used internally to process subsequent calls to NEXT(), PREVIOUS(), EOF() or BOF(). This handle may be used when calling the IDAPI functions directly from Clarion. Care must be taken when calling the IDAPI directly, since this can affect file processing by the driver.

SQL

```
s" = SEND(f, 'SQL=SELECT * FROM T') f{PROP:SQL} = 'SELECT * FROM T'  
s" = SEND(f, 'SQL') s" = f{PROP:SQL}
```

The SQL command or property is used to submit a query or determine the text of a previously submitted query. The SQL command may be used as a driver string on a file with no fields or keys to define a 'cursor'. A cursor provides a means of processing a table or tables according to some query defined at run-time. The results may be obtained using property syntax (see meta- commands). *This command is not available in the Standard variant of the driver.*

STRICT

```
s" = SEND(f, 'STRICT=1') f{PROP:Strict} = TRUE  
s" = SEND(f, 'STRICT') s" = f{PROP:Strict}
```

The STRICT command or property is used to enforce stricter checking to ensure that a file definition completely matches the table. Usually this is not necessary but it may be helpful during development and testing.

TABLECURSOR

```
s" = SEND(f, 'TABLECURSOR') s" = f{PROP:TableCursor}
```

The TABLECURSOR command or property is used to obtain the table's cursor handle. The cursor handle is used for all access to the table and is synchronised with the sequential cursor. This handle may be used when calling the IDAPI functions directly from Clarion. Care must be taken when calling the IDAPI directly, since this can affect file processing by the driver.

VERENGINE

i# = SEND(f, 'VERENGINE') i# = f{PROP:VerEngine}

The VERENGINE command or property is used to determine the version of the BDE that is in use.

VERINTERFACE

i# = SEND(f, 'VERINTERFACE') i# = f{PROP:VerInterface}

The VERINTERFACE command or property is used to determine the version of the BDE interface that is in use.

WORKDIR

s" = SEND(f, 'WORKDIR=C:\WORK') f{PROP:WorkDir} = 'C:\WORK'
s" = SEND(f, 'WORKDIR') s" = f{PROP:WorkDir}

The WORKDIR command or property is used to set or get the work directory for the current session. The work directory must be specified before the file is opened.

SQL Support

NOTE: The Professional variant of the dBASE File Driver supports SQL, QBE and Cursors as described in this section. These features are not available in the Standard variant of the driver.

The dBASE File Driver supports SQL commands using PROP:SQL or using a special form of FILE called a cursor (described below). If the SQL statement returns a result set, then a cursor must be used to retrieve the results.

The DBASE File Driver supports the IDAPI Local SQL dialect, this provides support for SELECT, INSERT, UPDATE and DELETE statements with some limitations.

Documentation on the IDAPI Local SQL dialect is included with Borland development tools (e.g. Borland C++ Programmer's Guide (Version 5), chapter 47.)

Cursors

The dBASE File Driver implements cursors using a special form of FILE declaration. A cursor traverses the result set of an SQL query. Furthermore, the query can be determined at run-time since a cursor can execute any query. A cursor is also used to process the results of a query using QBE.

A cursor is implemented using an empty file declaration. The query may be supplied as a driver string, or using the SEND() command or the PROP:SQL or PROP:QBE property:

```
!*** Cursor1 traverses table T
Cursor1      FILE,DRIVER('DBASE', 'SQL=SELECT * FROM T')
              RECORD
              END
              END

!*** Cursor2 must be determined using PROP:SQL or SEND('SQL=???')
Cursor2      FILE,DRIVER('DBASE')
              RECORD
              END
              END

CODE
OPEN(Cursor1)

Cursor2{PROP:SQL} = 'SELECT * FROM T'
OPEN(Cursor2)
```

Once the cursor is opened it may be traversed using the usual sequential file commands, SET(), NEXT() and PREVIOUS(). In addition any other file commands appropriate for sequential processing (e.g. POSITION(), RESET()) may be used. Note that cursors do not have keys, the order and content of the result set must be determined by the query.

A cursor is updateable if the result set refers to the original table data rather than a temporary table. The BDE will create a temporary table for the result set of a query satisfying any of the following conditions:

- It involves more than one table
- It has an ORDER BY clause

- It uses aggregates
- The WHERE clause does not consist only of simple comparisons of columns to constant values combined by AND or OR operators

If the result set is temporary, the cursor will be treated as read-only. Any attempt to update the cursor will result in an 'Access Denied' error.

Care must be taken when updating cursors on tables for which a primary key exists. If the primary key value is changed, the sequence in which records are returned in the result set will be affected. This can result in sequential loops (using SET() ... NEXT()) returning records in an unexpected order, processing records repeatedly or not at all, and possibly failing to terminate. As a general rule, do not modify primary key values.

The structure and contents of the record buffer must be determined using 'Meta File Properties', these are described in the accompanying document METAPROP.PDF.

Error Handling

The dBASE File Driver attempts to map any BDE errors to the appropriate Clarion error code. When it is not possible to do this, the driver will return an error 90. This indicates that driver specific error information is available via the FILEERROR() and FILEERRORCODE() functions. The FILEERROR() and FILEERRORCODE() functions may be called whenever any error is posted by the driver, to determine the IDAPI error.

Error codes in the range 2000h-2100h are returned to indicate a driver specific error, usually resulting from an illegal file structure. Error codes in the range 2100h-3F00h (and above) are IDAPI errors. The string returned by the FILEERROR() function gives a description of these errors.

During development it is advisable to check for errors in the above ranges after any OPEN() or CREATE() commands, to ensure that the file is properly formed.

Under certain circumstances the driver may display a 'Validation Error' dialog with a message like:

```
Error (n) validating registration data
```

This dialog is displayed when the driver fails when checking the registered user data or driver requirements at load time. The error number indicates the precise problem which may be one of the following:

- 0 - No error, however the driver evaluation has expired. You may download a later version or register the driver you have using the L3UNLOCK.EXE application.
- 1 - A consistency check on the registered user data failed. This indicates a corrupt driver that may have been tampered with.
- 2 - The key used to register the driver is incorrect. Should never occur.
- 3 - The Clarion IDE is not loaded. Unregistered copies of the driver require the Clarion IDE to be loaded.
- 4 - The .LIC file that was installed with the driver cannot be found. Unregistered copies of the driver require the .LIC file to be present on the PATH.
- 5 - An error occurred processing the .LIC file. This indicates a corrupt .LIC file that may have been tampered with.
- 6 - An error occurred validating the driver against the .LIC file. This may indicate a version mismatch between the .LIC file and the driver.

NULL Field Handling

The dBASE File Driver supports the SETNULL(), SETNONNULL() and NULL() commands though support for null fields in dBASE tables is limited and they should be used with care.

SETNULL() and SETNONNULL() operate by setting an internal flag to indicate whether the field is to be stored with a null value or not. As each record is read, the same flag is set for each field to indicate whether that field is null or not. The NULL() function simply tests the value of this flag, returning non-zero if the field is null, zero otherwise. All null fields are interpreted in Clarion as blank, as if initialised by the CLEAR() command.

In order to avoid problems with uninitialised DATE and TBDETimeStamp fields storing invalid values, the driver will store blank DATE and TBDETimeStamp fields as null.

FoxPro Support

The 32-bit dBASE File Driver includes support for FoxPro 2.0, 2.5 and 2.6 tables, indexes and memos. Since support is limited to the 32-bit driver and the Clarion IDE makes use of the 16-bit driver for browsing and Data Dictionary operations, it is necessary to use the DBF2TXD example utility when creating Data Dictionary entries for FoxPro tables.

When opening and accessing FoxPro tables the driver will automatically determine the correct format. However in order to create a FoxPro table it is necessary to use a driver string, SEND() command or property to force the correct format. For example:

```
FoxFile1  FILE, DRIVER('dBASE', '/DRIVER=FOXPRO')
          . . .
          END
```

```
FoxFile2  FILE, DRIVER('dBASE', '/LEVEL=25')
          . . .
          END
```

```
XbaseFile FILE, DRIVER('dBASE')
          . . .
          END
```

CODE

```
! Each of these examples will create a dBASE table
CREATE(xBaseFile)
```

```
xBaseFile{PROP:Level} = 5    ! Visual dBASE 5
CREATE(xBaseFile)
```

```
! Each of these commands will create a FoxPro table
CREATE(FoxFile1)
```

```
CREATE(FoxFile2)
```

```
xBaseFile{PROP:Driver} = 'FOXPRO'
CREATE(xBaseFile)
```

```
xBaseFile{PROP:Level} = 25
CREATE(FoxFile3)
```

The DBF2TXD utility will correctly identify the table type and create the correct data dictionary entries.

Transaction Support

The dBASE File Driver offers limited support for transactions. Transaction support is available only when using the BDE v3.0 and above and is restricted to 32-bit applications only.

Transactions on dBASE tables allow updates to be rolled back or committed, ensuring that applications perform updates in a consistent way. When a transaction is started by the LOGOUT() command, updates performed against dBASE tables are logged. Each log record contains the old record buffer of the record that is updated. When a transaction is active, the records with updates are locked and these locks are held until the transaction is either committed or rolled back.

The COMMIT() command releases all locks that were held whilst the transaction was active. The ROLLBACK() command restores the old values for each updated record in the table and then releases all the locks.

Inserted records are rolled back using soft deletes which means that although they are not visible they still present in the table. Use the PACK() command to remove deleted records.

There is no automatic recovery in the event of an application crashing during a transaction.

Utilities and Examples

The dBASE File Driver ships with some useful utilities and examples which are briefly described below.

DBInfo

The DBInfo utility must be shipped with any applications developed using the dBASE File Driver in order to comply with the licensing conditions for the Borland Database Engine (see Licensing the Borland Database Engine).

DBInfo is a simple diagnostic utility, developed using Borland C++ 5.0, which displays information about the BDE environment which may be of use in diagnosing problems with applications developed using the dBASE File Driver.

Full source for the DBInfo utility is supplied to allow for customisation and improvement.

DBF2TXD

The DBF2TXD utility is a hand-coded application that illustrates the use of MetaProperties to load and access tables. The utility is used to generate a TXD file from one or more dBASE tables that may subsequently be imported into a data dictionary. The DBF2TXD utility has several advantages over importing tables into the data dictionary using the Clarion IDE:

- More than one file may be imported at a time
- Properly augmented file descriptions are generated
- The 32-bit version can access dBASE 7 tables which are not accessible using the 16-bit IDE.

Technical Support & Resources

Should you experience any difficulties when using our software, you should take the time to read the chapters on **Unsupported or Modified Functions & Attributes** and **Limitations** to ensure that you are not trying to make use of an unsupported feature.

Visit the L3 Technologies web site (<http://www.l3.u-net.com>). You will find FAQ's, bug reports and workarounds and access to a variety of sources of information and other resources. Most importantly you will find the latest versions of the software for download.

If you are unable to resolve any issues using the information on the web site, you will find facilities for submitting a request for technical support.

Unsupported or Modified Functions & Attributes

This section lists unsupported functions and attributes and describes features or behaviour that differ in some respect from the norm. Should you experience any difficulties when using the driver, double-check this section to ensure that you are not trying to make use of an unsupported feature.

Where a feature is described as unsupported, an error will be posted if an attempt is made to use that feature. If a feature is described as ignored, its use will have no effect and no error will be posted.

- MEMO's are not supported. Consequently the NOMEMO() command is not supported. NOMEMO() does not apply to BLOBs which are loaded on demand.
- The BUFFER() command is not supported.
- dBASE tables do not support variable length records. Consequently the ADD(file, len), APPEND(file, len) and PUT(file, len) commands are not supported.
- Optimistic concurrency is not currently supported. Consequently WATCH() is not supported.
- The STREAM() and FLUSH() commands are ignored.
- The APPEND() and ADD() commands have identical functionality, i.e. the APPEND() command will update keys if any are defined.
- The DELETE() command simply marks the record for deletion, it will not be removed from the table until a PACK() is performed.
- The RECORDS() function returns the total number of records in the table, including those marked for deletion.
- Transactions (LOGOUT(), COMMIT(), ROLLBACK()) are supported by the 32-bit driver only, using v3.00 or above of the BDE.
- The OPT attribute is ignored.
- The PRIMARY attribute is supported for dBASE 7 tables only.
- The NOCASE attribute is ignored, however appropriate functionality can be achieved by using UPPER() or LOWER() in the keys expression to convert stored values to a single case.
- If the DUP attribute is omitted from a key or index declaration, the driver will still permit records with duplicate key values to be inserted into the table (unless this violates a dBASE 7 primary key), however only one entry will appear in the index itself. When processing the table in keyed order only one matching record will be retrieved.
- The RECLAIM attribute is ignored, though executing the PACK() command may reduce fragmentation and increase efficiency. BUILD() will achieve similar results.

Limitations

There are a number of limitations in this version of the dBASE File Driver which are listed below. The limitations may be removed in future versions of the driver.

- PROP:SQL can only issue a SELECT statement for a cursor file structure.

The following limitations are imposed by the current version of the Meta Property Extensions:

- Under Clarion for Windows 2.x there is no facility for declaring a key in a Meta File for use with commands which take a key (e.g. SET(key)).
- Cursors and dynamically created or loaded files cannot be binded.

Acknowledgements

All trademarks are trademarks or registered trademarks of their respective owners.