Class: DBF – open and manipulate a DBF file

Attributes:

CurRowDeleted	The row last read has been marked as deleted
Error	.true indicates an error has occurred – this instance should not be used and any changes made should be regarded as probably unmade.
ErrorText	An explanation of the error
FieldArr	an array of Field objects ordered as they appear in the DBF file. (see fields below)
Fields	A directory of Field objects accessible by name. (see below)
Fields[*field*]~various	Directory for use by subclasses
Index	Hook that user can hang an index on
ReturnType	STEM (default), DIRECTORY (fastest) or TABLE
Rownumber	The rowNumber of the last row read or appended.

File Header Attributes

These values stored in the file or field headers have two accessor methods ~text and ~value. ~Text is human readable, whilst ~value is usually the raw hexadecimal data as it appears in the file header. So no of rows can be accessed as dbf~items~text.

Attribute	~text	~value
CodePage	The code page associated with this file	1 hex byte
DataOffset	Where in file rows begin	2 hex bytes – little endian
Items	Number of rows in file	4 hex bytes – little endian
LastUpdate	When file last updated as ISO date, ie YYYY-MM-DD	3 hex bytes YYMMDD, where YY is no of years from 1900
Name	The table name. Ie: 'PRODUCT'	The name of the file. Ie: 'c:\product.dbf'
RecordLength	The length of each row including deletion marker	2 hex bytes little-endian
TableFlags	Special attributes of this file. One or more of: • file has a structural .cdx • file has a Memo field • file is a database (.dbc)	1 hex byte
Type	The type of dbf file from: • FoxBASE+/Dbase III plus, no memo • FoxBASE • Visual FoxPro • Visual FoxPro, autoincrement enabled • dBASE IV SQL table files, no memo • dBASE IV SQL system files, no memo • FoxBASE+/dBASE III PLUS, with memo • dBASE IV with memo • dBASE IV SQL table files, with memo • FoxPro 2.x (or earlier) with memo	1 hex byte An unknown type will cause an error

Field Header Attributes
These again have ~text and ~value accessor methods. They are accessed from a directory at fields. For instance to get the number of decimal places for a field called price you would use:
dbf~fields[‘PRICE’]~dps~text

Attribute	~text	~value
Dps	Decimal places (for numeric fields)	1 byte hex
Flags	Special attributes of this file, can be one or more of: • System Column (not visible to user) • Column can store null values • Binary column (for CHAR and MEMO only) • Column is autoincrementing	1 byte hex
Length	The length of this field	1 byte hex
Name	Field Name	Field Name
Next	For auto-incrementing fields, the next available value	4 byte hex – little endian
Step	For auto-incrementing fields, the step value	1 hex byte
Offset	Where the field starts in a dbf Row	4 byte hex – little endian
Type	Readable field type from {Character, Numeric, Float, Date, Logical, Memo, General, Double, Integer, Currency, DateTime, Picture, Unknown}	1 byte DBF field type indicator from {C, N, F, D, L, M, G, B, I, Y, T, P}
Reserved	The string 'Reserved'	The reserved area of the dbf file field header

NB Deleted Rows
Dbf files have a two-part deletion process. You can mark a row as deleted with the delete method or undeleted with the undelete method, but it is still there present in the file and any indexes and it is counted in the items attribute.

After fetching a row the attribute curRowDeleted will be set to .true if this is a deleted row.

Deleted rows are removed from the file by using the pack method at which point the record is removed physically from the file and the items attribute is updated.

NB: Currency of indexes
When you append to a file (unless you use the index append method), or update a key field (ie a field that an index is based on) or pack a file an index previously created will no longer be valid. It is the programmer’s responsibility to discard it or refresh it if required.
Indexes are created at create time and are not subsequently amended by append, delete, pack or update operations (except using the indexes own append or refresh method).

NB: Datasets
Several of the methods accept a dataset as input. The dataset may be a stem variable, table or directory whose indicies match the DBF field names (case sensitive).

Class Methods:

Create (**CLASS**) • Filename • Createstring	Creates and saves a new dbf file as specified by createstring (See dbf CreateStrings below).
New (CLASS) • Filename	Returns a new instance of the dbf class open for reading at the first record.

Instance Methods:

Append • Dataset Returns • Row Number	Appends a record to the dbf and populates it with the dataset. Returns the row number of the new record. NB: if an index is open use it's append method instead to update index.
Close	Closes the dbf file associated with this instance
CreateIndex • Definition • Options	Returns a dbfIndex object ordered by the definition. See Working with an index below. Options can include DELetes to include deleted rows in index.
Dataset • Field Data… Returns • Directory	Passed data in the order that fields are defined in the table, creates a directory object of field data indexed by field name ready for supplying to append or update.
Fetch • RowNumber	Returns a row specified by rowNumber as a dataset indexed by field name or .nil if no such record exists
FieldDelete • Fieldname	Creates a new copy of dbf without the specified field. NB: the DBF must be re-opened after each fieldDelete.
FieldExists • Fieldname	Returns .true if a field exists of that name.
FieldInsert • Fieldname • FieldType • FieldLength • [FieldDPs] • [ColNo] • [defValue]	Creates a new copy of the DBF with an added field. FieldNo may be an integer to indicate the desired placement of the field (defaults to last field). Data and deletion status is preserved for all extant fields/rows. If defValue is provided then the new field is populated with that in extant rows. NB: the DBF must be re-opened after each fieldinsert Return 0 – OK, -ve value error, +ve value windows error no
First	Returns the first record in the dbf file (sequential order) as a dataset indexed by field name or .nil if there are no records
Delete • RowSet	Marks all records in the passed set collection of row-numbers as deleted. The records will not be physically removed until a pack is done on the file. Rowset can be a single row number, or a set of rownumbers
Open	Re-Open a closed instance
Next	Returns the next record in the dbf file as a dataset indexed by field name or .nil if no such record exists
Pack	Physically removes any records marked for deletion and closes the file. Returns the number of records deleted or –1 on error.
Previous	Returns the previous record in the dbf file as a dataset indexed by field name or .nil if no such previous record exists
Undelete • RowSet	Marks all records in the passed set collection of row-numbers as not-deleted. Rowset can be a single row number, or a set of rownumbers
Update • Rowset • Dataset	Updates the field(s) referenced in dataset in the row(s) in rowset. Rowset may be a .set or a single number. Dataset indexes must match the case of the DBF fieldname (usually capitals). Missing dataset columns are ignored
Zap • Quiet Returns: • 0 – OK • -1, -2 – Error	Physically removes all records from file and closes file. Unless 'QUIET' is passed, the user will be prompted for confirmation. If the user cancels -1 is returned by the method If a file system error occurs –2 is returned

Examples of use

```
Dbf = .dbf~new('c:\test.dbf')
a. = dbf~first
do while a. \= .nil
   say '=== new row ==='
   do col over a.
      say col':' a.col
   end
   a.= dbf~next
end
dbf~close
```

deleting rows 1, 3 & 7

```
dbf = .dbf~new('c:\test.dbf')
dbf~delete(.set~of(1,3,7))
dbf~pack                                  /* closes the dbf */
```

Zapping a database

```
if .dbf~new('c:\test.dbf')~~zap('Quiet')~error
then say 'whoops'
else say 'whoopee'
```

Updating a field in selected rows

```
rows = .set~of(3,5)
drop a.
a.INSTOCK = 0
dbf = .dbf~new('c:\product.dbf')
dbf~update(rows,a.)
dbf~close
```

Returning a directory instead of a stem variable

```
Dbf = .dbf~new('c:\test.dbf')
Dbf~ReturnType = 'Directory' -- any caseless abbreviation
a = dbf~first
do while a \= .nil
   say '=== new row ==='
   do field over a
      say field':' a[field]
   end
   a = dbf~next
end
dbf~close
```

Dbf CreateStrings

Createstrings consist of fieldStrings seperated and suffixed by semicolons
Fieldstrings consist of fieldname, fieldtype, fieldlength & fieldprecision seperated by commas.
Here is an example of a valid createstring:
'SERIALNO,C,8,0;KEEPTILL,C,8,0;AMOUNT,N,6,2;SOLDDATE,C,8,0;'
example

```
schema = 'SERIALNO,C,8,0;KEEPTILL,C,8,0;AMOUNT,N,6,2;SOLDDATE,C,8,0;'
.dbf~Create('c:\mydbfs\items.dbf',schema)
```

Working with an Index

It is possible to create an index which is a view of the data ordered by chosen fields or parts of fields. Use the createIndex method of the dbf class to get a dbfIndex object.

Methods:

NB: All methods return a dataset on success and .nil on failure unless stated otherwise

Append Parameter: • dataset Returns • rowNo	Appends a record to the dbf and populates it with the values passed in the stem variable. Returns the physical row number of the new record, and inserts the record into the index.
Refresh Parameter • Dataset(s)	If a row with the same key exists it is updated, else a new row is appended. The number of rows updated is returned or –1 if an error occurs. Dataset may be an array of datasets
First	Returns the first record in the dbf file in the indexed order
Last	Returns the last record in the dbf file in the indexed order
Next	Returns the next record in the dbf file after the current record in the indexed order
Previous	Returns the previous record in the dbf file before the current record in the indexed order
Find Parameter: • Key	Returns the first record in the indexed order where key matches the index
FindGE Parameter: • Key	(Find Greater or Equal) Returns the first record in the indexed order where key matches the index. If no such record exists it returns the next record in the indexed order if such record exists
FindLE Parameter: • Key	(Find Less than or Equal) Returns the last record in the indexed order where key matches the index. If no such record exists it returns the previous record in the indexed order if such record exists
RefreshIndex	Will refresh the index after appends/deletes/updates performed on the dbf
Makekey Parameter • Dataset	Returns an index key (string). As Indexes can be complex, preparing a key to Find can be hard. Provide a dataset containing ALL the fields referenced in the index definition and Makekey will return the key.

The dbf class has an attribute index which can be used to store an index, meaning that the index can be accessed through the same variable that the dbf is. Indexes may take a lot of memory, and garbage collection won't collect them whist the dbf is in scope, so if you use the hook, you should release it when you are finished with the index (set hook to .nil).

Next and Previous

These methods can throw an error if the current record is outside the scope of the index – for instance current record is deleted. For this reason a NEXT or PREVIOUS on an index should always have been proceeded by a call to FIRST or LAST or a successful FIND, FINDGE or FINDLE. Note that FIND methods do not update the current record if they are unsuccessful.

Datasets

Datasets can be an ooRexx Directory, Table or Stem object. Which one the dbf class returns is governed by the ReturnType attribute. Where a dataset is passed to a method it doesn't matter which one is passed.

Index Definition Strings

Indexes contain elements which consist of fields or part of fields with an order (Ascending|Descending) and which Respect or Ignore Case. An Index Definition String can consist of one or more of these definition strings. Where there is more than one definition string, the first is the most significant, with the next only becoming significant in the case of equality.

Each definition for a string field is composed thus:
field,*start*,*length*,*order*,*case*;
All parts of the definition after the fieldname are optional and default like this:

Part	Default	Options
Start	1	Any start position within field
Length	Length of field	Any number of characters within field
Order	A	Ascending, Descending (any abbreviation)
Case	C	Case, Ignore, Numeric
Integers	All	
Decimals	All	

If a Start & Length is given then a sub-string of the field will be used
If Integers & Decimals are given then the numeric field will be formatted

Example Index Definition Strings

CODE	The index will order the table by the field 'CODE' in ascending order respecting case
CODE,8,1,D;CODE	The index will be sorted by the 8th Character of the field CODE in descending order. Where there is equality rows will be sorted by the entire CODE field in Ascending order.
SALEDATE,4,2	Date field ordered by month irrespective of year
SURNAME,,,,I	Ordered caselessly on surname
SURNAME;FORENAME;DOB	

NB: Index ordering is achieved using Prof. Rony Flatscher's StringColumnComparator from RGF_Util2. Read more about it at http://wi.wu-wien.ac.at:8002/rgf/rexx/orx20

Numeric fields

Fields with 0 decimal places in their definition are stored in dbf files right aligned without leading 0s and where appropriate prefixed by a '-' sign. This can make sorting as if the numbers were strings difficult. Using case 'N' will sort them by their numeric value. The start and length options are applied to the data before a comparison is made, so these can be used to truncate real numbers into integers, but care should be exercised using them any other way as one my inadvertently remove the sign of the data.

Performance

Indexes take memory. There is a trade off between key length and key complexity. In tests on a table with a key field (CODE) of 30 characters where only the first 8 were significant a 1.2% improvement in building and traversing an index with key 'CODE,1,8' was observed over one with key 'CODE'. Obviously this is affected by many factors including the amount of memory available on the target machine.

Examples of use

Working through data in a particular order (here indexed on field CODE)

DBF Helper classes

```
dbf = .dbf~new('c:\test.dbf')
ind = dbf~createIndex('CODE')
a. = ind~first
do while a. \= .nil
   Say a.CODE
   a. = ind~next
end
dbf~close
```

Retrieving all records after todays date

```
dbf = .dbf~new('c:\test.dbf')
dbf~index = dbf~createIndex('DATE')
a. = dbf~index~findGE(date('s'))
do while a. \= .nil
   Say a.DATE
   a. = dbf~index~next
end
dbf~close
```

Deleting all records more than a year old

```
dbf = .dbf~new('c:\test.dbf')
ind = dbf~createIndex('DATE')
a. = ind~findLE(date('s',date('b')-365,'b'))
deletes = .set~new
do while a. \= .nil
   deletes~put(dbf~RowNumber)
   a. = ind~previous
end
dbf~delete(deletes)
dbf~pack                     /* closes the dbf */
```

Matching particular record(s)

```
dbf = .dbf~new('c:\test.dbf')
ind = dbf~createIndex('SURNAME')
a. = ind~find('SMITH')
do while a. \= .nil & a.SURNAME = 'SMITH'
   Say a.forename a.surname
   a. = ind~next
end
dbf~close
```

Appending a record with index open

```
dbf = .dbf~new('c:\test.dbf')
ind = dbf~createIndex('SURNAME')
…
drop a.
a.SURNAME = 'SMITH'
a.FORENAME = 'Sally'
ind~append(a.)
dbf~close
```

Using the index Hook

```
dbf = .dbf~new('c:\test.dbf')
dbf~index = dbf~createIndex('SURNAME')
…
drop a.
a.SURNAME = 'SMITH'
a.FORENAME = 'Sally'
dbf~index~append(a.)
dbf~index = .nil /* allow garbage collection to get index */
                 /* if you no longer need it              */
…/* other dbf operations that don't need the index */
dbf~close
```