FlagShip

Object Oriented Database Development System

Cross-Compatible to Unix, Linux and MS-Windows

MULTISOFT

Release 8.1

Section RDD

The whole FlagShip 8 manual consist of following sections:

Section	Content
GEN	General information: License agreement & warranty, installation and de-installation, registration and support
LNG	FlagShip language: Specification, database, files, language elements, multiuser, multitasking, FlagShip extensions and differences
FSC	Compiler & Tools: Compiling, linking, libraries, make, run-time requirements, debugging, tools and utilities
CMD	Commands and statements: Alphabetical reference of FlagShip commands, declarators and statements
FUN	Standard functions: Alphabetical reference of FlagShip functions
OBJ	Objects and classes: Standard classes for Get, Tbrowse, Error, Application, GUI, as well as other standard classes
RDD	Replaceable Database Drivers
EXT	C-API: FlagShip connection to the C language, Extend C System, Inline C programs, Open C API, Modifying the intermediate C code
FS2	Alphabetical reference of FS2 Toolbox functions
QRF	Quick reference: Overview of commands, functions and environment
PRE	Preprocessor, includes, directives
SYS	System info, porting: System differences to DOS, porting hints, data transfer, terminals and mapping, distributable files
REL	Release notes: Operating system dependent information, predefined terminals
APP	Appendix: Inkey values, control keys, ASCII-ISO table, error codes, dBase and FoxPro notes, forms
IDX	Index of all sections
fsman	The on-line manual “f**sman**” contains all above sections, search function, and additionally last changes and extensions

multisoft Datentechnik, Germany

Object Oriented Database Development System,
Cross-Compatible to Unix, Linux and MS-Windows

Section RDD

Manual release: 8.1

For the current program release see your Activation Card,
or check on-line by issuing *FlagShip -version*

Note: the on-line manual is updated more frequently.

Copyright

Made in Germany. Printed in Germany.

Trademarks

Headquarter Address

multisoft Datentechnik
Schönaustr. 7
84036 Landshut
Germany

Phone: *(*+49) 0871-3300237

E-mail: support@flagship.de
support@multisoft.de
sales@multisoft.de

Web: http://www.fship.com

RDD: Replaceable Database Drivers

1. The basics of RDDs

FlagShip fully supports the Replaceable Database Driver (RDD) architecture. This means, you may use another (e.g. your own) RDDs instead, or in coexistence with the FlagShip standard database driver named "DBFIDX".

The following description generally assumes, you are already familiar with the basics of FlagShip or Xbase programming (section LNG) and often refers to standard database commands (section CMD), functions (section FUN) and objects (section OBJ). Please read these chapters first.

1.1 Why Different Database Drivers?

Although the database file (.dbf) is common for a wide range of Xbase systems (FlagShip, VO, Clipper, FoxBase, FoxPro, dBase), the internal structure of memo files (.dbt, .fpt) may vary between the systems (e.g. FoxPro and dBase IV use their own structures). The most significant difference and incompatibility is noticeable on the index structure and the used locking algorithm.

As long as you use a single database system, you don't need to care about the internal file structure, and about the internal algorithms used to interchange information, if and how the database is locked. All professional database systems are smart enough to manage this information automatically also in a local network, some of them (like FlagShip) also in wide area and heterogeneous networks.

When you need access (or even need the simultaneous access) to databases and indices, created from, and also managed by an other Xbase system, you need a mechanism to simulate this access, an interface to the "other" system. Also, if you need to manage a non-Xbase databases, such as SQL (or ASCII files) are, you do not have to access them at the low level via C/ESQL, or via FREAD(), but may handle them as "usual" databases. All these interfaces are realized by the Replaceable Database Driver RDD.

Since the database driver is tied to a working area, you may theoretically use up to 65,000 different drivers, each supporting another kind of database. On the other hand, if only one driver is used, up to 65,000 databases of the same type may be open simultaneously.

1.2 Working Areas, Databases

The FlagShip database system (including the database and index access commands and functions) is designed around the working area. It specifies slots, each of which can handle one database (also called a table) with several memo files, indexes and relations at a time. FlagShip supports up to 65534 such slots (working areas). You may select the working area

explicitly by the SELECT command and DBSELECTAREA() function, or implicitly via the NEW clause in the USE command or DBUSEAREA() function.

A working area is occupied or unoccupied, depending on whether a file is opened in it. At program startup, all areas are unoccupied and the slot (work area) one is the current.

A database (table) consists of a variable number of rows, called records. Each record has the same number of columns, called fields. Each field is identified by a unique name within the table. The RDD knows the name, type and the size of each field, usually stored in the database header.

The supported size of the table depends on the RDD used. The default FlagShip's RDD named "DBFIDX" supports up to 2 billion records, each with up to 64000 fields, or up to 2 Gbyte of size.

Each working area manages (at run-time) its own pointer to the current record and the access to fields of this record in the same way as access to a variable, or through field access functions and methods. You move the record pointer (also named "cursor" in some systems) explicitly by e.g. GOTO, SKIP and DBGOTO(), or implicitly via SEEK, relation movement etc. See additional details in sections LNG, CMD, FUN and OBJ.

1.3 The Architecture of RDD

In FlagShip, each Replaceable Database Driver is independent of other RDDs. The only communication between the RDD and the application is performed via the RDD interfaces, namely the object methods (and exported instances).

You will access the standard or selected RDD in the current (or new) working area by invoking the standard command USE, the function DBUSEAREA(), or by instantiating the RDD object itself.

The RDD stores the internal **data** (instances) of the object (the database) in the corresponding working area slot, but always uses the same, common program **code**. In other words: the program code of the RDD is loaded once only, whilst each occupied working area manages only its own, internal data.

FlagShip fully supports the hybrid (intermixed) usage of all standard database commands, functions and object methods. So it is your choice, to use the Xbase common commands like USE, SKIP, GOTO, INDEX, SEEK etc, the Clipper-like functions DBUSEAREA(), DBSKIP(), DBGOTO(), DBCREATEINDEX(), DBSEEK() etc., or the RDD object methods oRdd:SKIP(), oRdd:GOTO(), oRdd:SEEK() etc. Each of these programming techniques is described in detail in sections CMD, FUN or OBJ, respectively.

In fact, all the commands (e.g. SKIP) are translated by the preprocessor to high-level database functions (e.g. DBSKIP()) according to #command directives specified in the std.fh file. These database functions call then the equivalent RDD method (e.g. oRdd:SKIP()) of the driver,

associated with the current working area. Some commands or functions may access several RDD methods.

The same is valid for an access of database fields. You may access them directly by name, by using the FIELGET() function, or via the oRdd:FIELDGET() method.

In some cases, a database command or function acts globally on all open database files (for example, CLOSE ALL or DBCOMMITALL()), or access two different RDDs at a time (e.g. COPY TO...VIA...), and is therefore a superset of the underlying RDD method. In other cases, some RDD methods provide you with more flexibility and information about the driver (e.g. oRdd:INFO()), not available as a high-level database function.

Usually, the high-level commands and functions are preferred by most programmers, because of simple notation, clearer syntax and higher degree of portability, whilst the RDD methods are invoked for special information or actions only. In FlagShip (as opposed to some other systems), there is no significant difference in the execution speed between the high-level and the method invocation (assuming, the RDD invocation uses prototypes for an early binding).

For programmers, who wish to build their own RDDs, details are given in chapter RDD.2.

1.4 Choosing an RDD

To use an RDD, the corresponding driver has to be linked in with your application. You can then select and activate the driver in several ways.

The default database DBFIDX driver, (which supports the .dbf, .dbt and .idx files), is available in the FlagShip library. Usually, this driver is linked automatically via the RddInit procedure, which is also included in the library. Since this is an INIT procedure, it is invoked automatically at the application startup.

You may freely modify the behavior of the RddInit procedure, e.g. to disable the automatic DBFIDX linking (to shrink the application size, when it does not handle database access at all), or to predefine another (e.g. 3rd party) or additional RDD driver(s) to be linked automatically. The source code of RddInit is available in the <FlagShip_dir>/system/rddsys.prg file.

If you need to change this file, the best method is to copy it into your local directory, make the changes, compile it separately according to the instructions in the file header, and then simply link the object file (rddsys.o) with your application.

Of course, it is not required to change the rddsys.prg file, whenever you need to use an additional RDD driver. Instead, a simple statement EXTERN <rddName>NEW (or REQUEST <rddName>NEW) somewhere in your program code does the same and will also link the <rddName> driver of your choice.

To select an RDD to be used by default for all subsequent USE commands and DBUSEAREA() functions, regardless of the working area, identify the (linked) RDD via the RDDSETDEFAULT() or DBSETDRIVER() function. Since the DBFIDX driver is already set to default in the RddInit function, you do not need this to use it.

You may also specify another RDD (than the default), which should explicitly be used in the current (or new) working area. To do so, identify the driver using the VIA clause of the USE command, or the equivalent parameter of the DBUSEAREA() function. Of course, instantiating the selected RDD object does the same.

1.5 Relationships between RDDs

As stated earlier, each RDD is tied to the occupied working area. Many database commands and functions support access to RDDs, other than the current one. For example, you may insert records into the current (e.g. a .dbf) database from other table types (e.g. SQL) by simply using the VIA clause of the APPEND FROM command. In such a case, the "other" driver reads the records, while the local driver adds them into the current database. This is similar to reading fields from one database (into memory variables), switch the working area, and replace the other database or table through those. Also, using relations to databases, driven by other RDDs is possible.

Of course, such a tight coexistence of two (or more) RDD drivers requires, that both drivers support the same data types and own the standard methods for such data exchange. Also, both have to be inherited from the DataServer class.

For simple read/write operations on the current database, the RDD may be simplified and can support only a subset of the standard commands and functions. If so, you will receive a run-time error "method ... not available" if you try to invoke an unsupported feature of the RDD. To avoid the RTE (run-time-error), you may check the availability of methods (e.g. in a unknown RDD) by using the ISOBJPROPERTY() function.

1.6 Invoking the RDD

The RDDs may be supplied in the standard FlagShip library (e.g. DBFIDX), in additional, user or 3rd party libraries (e.g. CB4CDX), in object form (*.o), or in source code.

You may link the RDD supplied in object code directly with your application, for example
$ FlagShip myappli*.prg myRdd*.o

The same applies, if the RDD is supplied in source code. You will first compile the RDD sources according to the supplied instructions, and then link the object code with your application.

If the RDD is available in (any) library, you have to tell the linker, to search the library for the required modules. The simplest method is to use the EXTERN or REQUEST statement, as described in chapter 1.4. Of course, this may also be performed automatically in an INIT procedure or function (e.g. the RddInit). Then, link the corresponding library (e.g. the libMyRdd.a) by

```
$ FlagShip myappli*.prg -L/usr/mylibs -lMyRdd
```

It is of course not necessary to explicitly specify the standard FlagShip library (<FlagShip_dir>/libFlagShip*.a) in the command line to use default RDD drivers, since this library is used automatically.

In the application, the RDD is then invoked automatically through the VIA clause of the USE or APPEND FROM etc. command, the corresponding parameter of the equivalent standard function, or explicitly by instantiation of the RDD via the oRdd := rddName{...} or oRdd := rddNameNEW(...) statement.

1.7 Example of RDD usage

The following example demonstrates the usage of different RDDs and the hybrid programming techniques. Many additional examples are given in sections LNG, CMD, FUN and OBJ.

```
#include "fspreset.fh"                 // use files in lower case
#include "dbfidx.fh"                   // prototypes of the DBFIDX class
#include "cb4cdx.fh"                   // prototypes of the CB4CDX class
#include "rddsys.fh"                   // constants or RDDs used
LOCAL ok AS LOGICAL
LOCAL oRdd1 AS Object, oRdd2 AS DbfIdx, oRdd3 AS Cb4Cdx

SELECT 597
USE Address INDEX adr1, adr2 ALIAS addr
if !used() .or. NetErr()
  ? "sorry..."
endif
oRdd1 := DBOBJECT()                    // typed for late binding
? SELECT(), ALIAS(), oRdd1:ALIAS       // 597 ADDR ADDR

// USE Article Index Article NEW SHARED VIA DbfIdx
if (ok := DBUSEAREA (.T., "DbfIdx", "Article", NIL, .T., .F.))
   ok := DBSETINDEX("Article")
endif
if !ok
   ? "sorry..."
endif
oRdd2 := DBOBJECT()                    // typed for early binding
? SELECT(), ALIAS(), oRdd2:ALIAS       // 1 ARTICLE ARTICLE

EXTERN Cb4CdxNEW                       // force to link it

// USE FoxData NEW SHARED VIA ("Cb4Cdx")
oRdd3 := Cb4Cdx {"FoxData", .T., .F., NIL, NIL, .T.}
if !oRdd:USED
   ? "sorry..."
endif
oRdd3:ALIAS := "FoxAli"
? SELECT(), ALIAS(), oRdd3:ALIAS       // 2 FOXALI FOXALI

? oRdd1:INFO(DBI_ACCESSRIGHTS), oRdd2:INFO(DBI_ISDBF), ;
  oRdd3:RDDINFO(_SET_MEMOEXT), oRdd3:INFO(DBI_MEMOBLOCKSIZE)
CLOSE DATABASES
```

2. Writing New RDDs

This chapter handles the basics for programming Replaceable Database Drivers. You may skip to chapter 3, if the programming of RDDs is not relevant for you.

2.1 Structure of the RDD

As mentioned in chapter 1, all RDDs in FlagShip are encapsulated objects. The RDD methods perform all the required database actions, independent from the rest of the application. The programming follows the FlagShip OOP programming rules, described in section OBJ. Your programming language is FlagShip (.prg), Extend and Open C API (.c), or a combination of both.

It's your choice, to specify your own CLASS for the RDD, or inherit your class from the default DataServer (or DbfIdx) class. The user specific class may sometimes be smaller, or more suitable, than the inherited one.

The advantage of the inherited classes is, that all standard methods (including Access and Assign) are already predefined. In the DataServer class, all of them are empty, but point to functions, similar to NoiVarGet(), NoiVarPut() and NoMethod() object methods. This may simplify your object significantly.

All these techniques are demonstrated in the supplied source code files. The "smallrdd.prg" is a small driver for .dbf based databases, inheriting the DataServer class and fully written in .prg language. The rddcb4*.* sources demonstrates hybrid programming of .prg and .c language, whereby its class also inherits the DataServer one. The "ascirdd.c" RDD performs a special, limited functionality to (read only) handle ASCII files. It also inherits DataServer class (and announces the fields to FlagShip), in order to enable the FOR/ WHILE scoping on, and named access to the field variables.

2.2 FlagShip Communication with RDDs

From the view of an application programmer, the only direct communication with the RDD is via the supplied properties (methods, exported instances) of the RDD. As stated earlier, he may also use the standard FlagShip commands and functions instead; these high-level functions then manage the messages sent to the RDD.

From the view of the RDD programmer, there are three independent pre- requirements, which should be met for the full support of the hybrid (or high level, procedural) programming:

a. The invocation of the RDD methods from FlagShip database high-level commands and functions (e.g. via SKIP, DBGOTO() etc.) is only possible, if the RDD class inherits the

general DataServer class, or any child (e.g. DbfIdx) thereof. This is because the FlagShip functions use an early binding, which can properly work on a known class only.

b. The RDD must announce the occupying and freeing of a working area to FlagShip, when the object is instantiated or closed (usually in the INIT and CLOSE method). Otherwise, you have to avoid using the USE command and the DBUSEAREA() function in the same working area. This is because the working areas (and their occupying) are managed by FlagShip (e.g. by SELECT), not by the RDD itself.

c. To be able to use field names in the same way as memory variables, the RDD must announce these names to FlagShip, usually done in the INIT method. Also, the CLOSE or AXIT method has to retire this announcement. Otherwise, the FIELDGET() and FIELDPUT() functions and/or methods have to be used instead.

If one of these pre-requirements is not met by the RDD, only partial hybrid programming is possible. If none is met, only the object invocation of the RDD is possible.

Additionally, the RDD may contain

d. an INIT procedure or function (of any name, e.g. INIT PROC _rddName) which calls AnnNewRdd("rddName") to announce the driver name for the standard RDDLIST() function.

2.3 Required RDD methods

The RDD must supply the INIT, CLOSE and AXIT methods, and should mostly also support at least FIELDGET and USED, see also LNG.2.11.3 and OBJ.1.2.

2.3.1 Init() Method

The INIT method of the RDD is called automatically from the object creator function rddNameNEW, which is invoked during the object instantiation, or from the DBUSEAREA() function. All parameters from rddNameNEW(...) are passed to the INIT method. It has to perform all required parameter checking, initialize the instances, open the given file name (with the associated memo files, if any) and return the object SELF.

For hybrid usage, the INIT method may announce the occupancy of the current working area to FlagShip, see 2.2.b. To do it, invoke the (.prg callable) function

```
cNewAlias := RDDannAlias (oSelf, cSuggAlias)
```

where <oSelf> is the object SELF, <cSuggAlias> is the new Alias name desired and <cNewAlias> is the Alias name granted and used by FlagShip. RddannAlias() will confirm the passed <cNewAlias> in the returned <cNewAlias>, or will build it from the supplied file name. If the resulting alias already exists, a new unique name is created.

You may use the TRUEPATH() standard function to adopt the supplied file name according to the current SET PATH and FS_SET() settings, if required.

To announce the field names to be usable as usual variables (see 2.2.c), create an empty array (for internal FlagShip and for FIELDGET() use), the size of which is at least the number of passed field names. Assign this array to a STATIC variable or an object instance. Then invoke for each used field the function

```
lOk := RddAnnField (cName, nPos, cType, iAccess, nLength, nDeci, aMyArr)
```

where <cName>, <cType>, <nLength> and <nDeci> describes the field according to DBSTRUCT(), <nPos> is the consecutive field number, <iAccess> specifies the access type, and <nMyArr> is the internally used array (which will contain pointers to corresponding field variables). The <iAccess> is either 0 or 1. Zero specifies, that all fields and the corresponding field variables are updated by the RDD on every record movement. Specifying One allows an optimized record access, it announces FlagShip to invoke the oRdd:FIELDGET() method only if the field value is required.

This all may sound a little bit complicated, but is quite easy, see example in chapter 2.4 below and in the smallrdd.prg file, available in <FlagShip_dir>/system/smallrdd. For a C invocation, see example in the rddcb4*.c files, available in <FlagShip_dir>/system/cb4rdd directory.

2.3.2 Close() Method

The CLOSE method of the RDD is automatically invoked only on hybrid usage from the USE, CLOSE DATABASE and CLOSE ALL commands or equivalent functions, and at the termination of the application. The automatic invocation requires at least the announcing of the working area occupancy, according to 2.2.b. A forced invocation of this method is also opportune from the AXIT method.

The CLOSE method should reverse the announcement of occupying the working area and the database fields, which is done in the INIT method. First, free the announced fields by invoking

```
RDDretField (SELF, cName)
```

for all announced fields in INIT. Then free the working area by

```
RDDretAlias (SELF)
```

Note, that this statement may also destroy the object (if no other object variable additionally refers to it) and is therefore the last valid operation on the instances.

2.3.3 Axit() Method

Normally, the AXIT method, if available in an object, is invoked automatically by the FlagShip variable system, if the lifetime of the carrying object expires (e.g. for a LOCAL object at the end of UDF). AXIT may return any value, except SELF.

With an RDD however, even if the LOCAL variable lifetime expires, the carrying object is not destroyed automatically, if the RDD was announced to FlagShip via RDDannAlias(). In such a case, the object is not destroyed before explicitly invoking oRdd:CLOSE(), CLOSE or DBCLOSEAREA(). You may therefore regain access to an object of an occupied working area by the means of DBOBJECT().

The AXIT method should free all manually allocated memory space (e.g. malloc() assigned to a SPECIAL variable via _xalloc() etc). All usual FlagShip variables are freed automatically by the FlagShip garbage collector.

2.3.4 FieldGet() Method

The FIELDGET (or QUICKFIELDGET, if available) method allows to access fields of the current selected RDD record. The method receives the consecutive field number (starting by one), or optionally the field name, and returns the value of the field.

On hybrid usage, this method should also fill the FIELD variable to allow the application to use fields in the same way, as usual memory variables, see also 2.2.c. To do so, invoke

```
VarPtr := AssignFldValue (@xMyArrElem, value)
```

where <xMyArrElem> is an element of the in INIT by RddAnnAlias() allocated array, and <value> is the field value, already converted to the expected variable (field) type. The function returns the pointer to the field variable, available in the application.

The FIELDGET() method should return the field variable. The best is to use RETURN @AssignFldValue(...). The difference to the usual RETURN <value> is, that the memory variable <value> would not contain the type "M" but "C", and numerics would be displayed in variable, rather than fixed length.

2.3.5 Access Method USED

This access method is generally used after the instantiation (or in DBUSEAREA()) to determine, if the RDD is usable, i.e. if the database was open successfully. See example below in chapter 2.4.

2.4 Example of minimal, hybrid RDD for SDF

The following example is a fully usable RDD driver for ASCII files in SDF format. To avoid the overhead for the purpose of this overview, only very minimal parameter checking is performed and only forward skipping is supported. As you may also conclude by yourself, additional performance optimizing and validity checks of the file are possible. For a more featured RDD, see the <FlagShip_dir>/system/smallrdd/smallrdd.prg program file. For RDD written in C, inspect the <FlagShip_dir>/system/ascirdd/ascirrd.c and <FlagShip_dir>/system/cb4rdd/cb4cdx?.c files.

```
* file minirdd.prg

#include "dataserv.fh"                          // DataServer prototype
#include "dbstruct.fh"                          // defines DBS_xxx

CLASS MyRdd INHERIT DataServer
   PROTECT aFieldVars    AS Array               // ptr's to field vars
   PROTECT aStruct       AS Array               // suppled structure
   PROTECT iHandle := 0  AS IntVar              // Fopen() handle
   PROTECT iRecBeg := 0  AS IntVar              // posit of rec begin
   PROTECT iRecLen := 0  AS intVar              // record length
   PROTECT lUsed := .F.  AS Logical             // RDD usable ?
   EXPORT  myAlias       AS Character           // alias
   EXPORT  myEof := .F.  AS Logical             // eof ?

METHOD INIT (cName, aDbStru) CLASS MyRdd        // note spec. parameters,
   Local ii AS intVar                           // different from std.
   if valtype(cName) != "C"                     // USE and DataServer
      return SELF
   endif
   iHandle := FOPEN(cName, 0)                   // open the file r/o
   if iHandle <= 0 .or. len(aDbStru) = 0
      return SELF                               // error, not usable
```

```
   endif
   aStruct := ACLONE (aDbStru)
   myAlias := RDDannAlias (self, cName)       // Announce WA to FS
   aFieldVars := ARRAY(len(aDbStru))          // used in FieldGet()
   for ii := 1 to len(aDbStru)                // Announce FIELD Vars:
      RDDannField (aDbStru[ii,DBS_NAME],  ;   //  field name
                   ii,                    ;   //  field pos
                   aDbStru[ii,DBS_TYPE],  ;   //  field type
                   1,                     ;   //  late field access
                   aDbStru[ii,DBS_LEN ],  ;   //  field length
                   aDbStru[ii,DBS_DEC ],  ;   //  field deci length
                   aFieldVars)                //  FIELD vars ptr
   next
   lUsed := .T.                               // assign instance
   return SELF

METHOD Close CLASS MyRdd
   Local ii AS int
   if iHandle = 0 .or. !lUsed                  // file open?
      return .F.
   endif
   FCLOSE (iHandle)                           // close file
   lUsed   := .F.                             // reset instances
   iHandle := 0
   for ii := 1 to len(aFieldVars)             // free FIELD vars
      RDDretField (self, aStruct[ii,DBS_NAME]) // in FlagShip
   next
   RDDretAlias (self)                         // free WA in FS
   return .T.

METHOD Axit CLASS MyRdd                       // here: forces also
   if iHandle > 0 .and. lUsed                 // closing the file
      SELF:Close()                            // to be able free
   endif                                      // the object
   return .T.

ACCESS METHOD Used CLASS MyRdd                // is the RDD usable?
   return lUsed

METHOD FieldGet (nPos) CLASS MyRdd
   Local value, ii, iFldPos := 0, buff, iLen, cType
   if !lUsed .or. iHandle <= 0 .or. nPos <= 0 .or. nPos > len(aStruct)
      return NIL
   endif
   iLen := aStruct[nPos,DBS_LEN]
   cType:= upper(left(aStruct[nPos,DBS_TYPE],1))
   for ii := 1 to nPos -1
      iFldPos += aStruct[ii,DBS_LEN]          // determine field begin
   next
   buff := space(iLen)
   FSEEK (iHandle, iRecBeg + iFldPos, 0)      // posit to field begin
   FREAD (iHandle, @buff, iLen)               // read flat ASCII file
   if cType == "C"
      value := buff
   elseif cType $ "NIF"                       // convert ascii to var
      value := val(alltrim(buff))             // accord to field type
```

```
   elseif cType == "D"
      value := stod(buff)
   elseif cType == "L"
      value := upper(left(buff,1)) $ "YT"
   else
      value := NIL
   endif
   return @ AssignFldValue(@aFieldVars[nPos], value)

METHOD Skip (nRec) CLASS MyRdd                  // very simplified
   Local  ii AS int                             // for flat ASCII file
   if iRecLen == 0                              // in SDF format
      for ii := 1 to len(aStruct)               // Determine the
         iRecLen += aStruct[ii,DBS_LEN]         // record size
      next
      iRecLen ++                                // assumed LF byte,
   endif                                        // but should be checked
   myEof := .F.
   for ii := 1 to abs(nRec)                     // here forward only!
      FSEEK (iHandle, iRecBeg + iRecLen, 0)     // posit of the first
      iRecBeg := FSEEK(iHandle, 0, 1)           // field per record
      if iRecBeg >= FSEEK (iHandle, 0, 2)       // EOF() ?
         myEof := .T.                           // set export inst.
         exit
      endif
   next
   return !myEof

INIT PROCEDURE _myRdd                           // for RDDLIST()
   AnnNewRdd ("myRdd")                          // purpose only
   return

* ----------------- Main (test) program -----------------------------

FUNCTION start()
* extern myRddNEW                               // not required here
Local aStru, oRdd
if !file("minirdd.txt")                         // create the file
   ? "Creating SDF ascii file:" ; ?             // used below
   set printer to minirdd.txt
   set printer on
   ?? "A1aaaaaaaA 1       B1bbB19950101"
   ?  "A2          2.2    b2   00020202"
   ?  "A3aaaaaaaA3333333.3B3bbB19951231"
   ?                                            // last LF expected
   set printer off
   set printer to
endif

astru := {{"f1","c",10,0}, {"f2","n",9,1}, ;
          {"f3","c", 5,0}, {"f4","d",8,0}}

oRdd:= myRdd {"minirdd.txt", astru}             // instantiate RDD
if !oRdd:used
   ? "Cannot open ASCII database 'minirdd.txt'"
   quit
```

```
endif
? "Select, Used, Alias=", select(), used(), oRdd:myAlias

while !oRdd:myEof
   ? f1, f2, f3, f4, " == ", oRdd:FieldGet(1), oRdd:FieldGet(2)
   skip 1
enddo
use
use minirdd.txt via MyRdd
if !used()
   ? "Cannot invoke MyRdd via USE ..."
   ? "  This is an expected error, since myRdd:INIT() parameters"
   ? "  do NOT match those of DataServer:INIT() passed from USE"
endif
quit

* eof
*** Compile: FlagShip minirdd.prg -na -Mstart
```

2.5 RDD methods used in std. functions

The following cross reference demonstrates the dependence standard FlagShip database functions of the RDD methods. Note, that this is for your information purposes only, and may be changed without notice.

FS function	used RDD method or standard database functions
__DbApp()	DbEval(), DbSelectAr(), DbStruct(), Fcount(), FieldPos(), NetErr(), elect(), Used()
__DbAppDel()	rdd:AppendDelimited()
__DbAppSdf()	rdd:AppendSDF()
__DbContin()	DbEval(), DbSkip(), Found(), Used()
__DbCopy()	__DbCopySt(), Afields(), Alias(), DbEval(), DbSelectAr(), DbUseArea(), Fcount(), RddName(), Select(), Used()
__DbCopyDe()	Afields(), DbEval(), Used()
__DbCopySd()	Afields(), DbEval(), Used()
__DbCopySt()	DbCreate(), DbStruct(), FieldPos(), Used(), rdd:Info(DBI_FILEHANDLE)
__DbCopyXs()	__DbCreate(), DbCloseAre(), DbSelectAr(), DbStruct(), DbUseArea(), Select(), Used(), rdd:Info(DBI_FILEHANDLE)
__DbCreate()	DbCloseAre(), DbSelectAr(), DbSkip(), DbUseArea(), Eof(), NetErr(), RddName(), rdd:Info(DBI_FILEHANDLE)
__DbList()	Afields(), DbEval(), Deleted(), RecNo(), Used()
__DbLocate()	DbEval(), Found(), Used()
__DbPack()	rdd:Pack()
__DbSort()	__DbCopy(), DbCreateIn(), DbSelectAr(), DbUseArea(), FieldBlock()
__DbTotal()	Alias(), DbAppend(), DbCloseAre(), DbCreate(), DbGoto(), DbGotop(), DbSkip(), DbStruct(), DbUseArea(), Eof(), FieldGet(), FieldPut(), Select(), Used(), rdd:Info(DBI_FILEHANDLE)
__DbUpdate()	Alias(), DbSkip(), Eof(), Found(), Seek(), Select(), Used()
__DbZap()	rdd:Zap()
_DbFind()	DbSeek(), rdd:OrderInfo(DBOI_KEYTYPE)
_DbJoin()	Afields(), Alias(), DbCloseAre(), DbCreate(), DbEval(), DbSelectAr(), DbSkip(), DbStruct(), DbUseArea(), Eof(), Fcount(), FieldPos(), FieldWblock(), NetErr(), Select(), rdd:Info(DBI_FILEHANDLE)
_SeekEval()	rdd:SeekEval()
AFields()	rdd:Fcount[acc], rdd:FieldName(), rdd:FieldInfo(DBS_LEN), rdd:FieldInfo(DBS_DEC), rdd:FieldInfo(DBS_TYPE)
Alias()	rdd:Alias[acc]
AutoxLock()	rdd:Commit(), rdd:Flock() or rdd:Rlock(), rdd:Used[acc], Dbf() or rdd:Info(DBI_FULLPATH),
Bof()	rdd:Bof[acc]
DbAppend()	rdd:Append()
DbClearFil()	rdd:ClearFilter()
DbClearInd()	OrdListCle()
DbClearRel()	rdd:ClearRelation()
DbClose()	rdd:Close()

DbCloseAre()	rdd:Close()
DbCommit()	rdd:Commit()
DbCommitAl()	rdd:Commit(), rdd:RecNo[acc/ass], rdd:Relation[acc/ass]
DbCommitAl()	rdd:Commit()
DbCreate()	rdd:CreateDb()
DbCreateIn()	OrdCreate()
DbDelete()	rdd:Delete()
DbEdit()	Bof(), DbGoBottom(), DbGoto(), DbGoTop(), DbSkip(), DbStruct(), Eof(), IndexOrd(), IsDbExcl(), LastRec(), Recno(), Select(), Used()
DbEval()	rdd:Eval()
Dbf()	rdd:FileSpec[acc]
DbFilter()	rdd:Filter[acc]
DbGetLocat()	rdd:GetLocate[acc]
DbGoBottom()	rdd:GoBottom()
DbGoto()	rdd:GoTo()
DbGoTop()	rdd:GoTop()
DbObject()	
DbRecall()	rdd:Recall()
DbReindex()	OrdListReb()
DbRelation()	rdd:Relation()
DbRlockLis()	rdd:RLockList[acc]
DbRselect()	rdd:RelationObject()
DbRselect()	rdd:SetRelation()
dbRunlock()	rdd:unlock()
DbSeek()	rdd:Seek()
DbSelectAr()	rdd:Used[acc]
dbSetDefa()	
DbSetFilte()	rdd:Filter[ass], rdd:FilterString[ass]
DbSetIndex()	OrdListAdd()
DbSetLocat()	rdd:Info(DBI_GETSCOPE)
DbSetOrder()	OrderSetFocu()
DbSetRelat()	rdd:SetRelation()
DbSkip()	rdd:Skip()
DbStruct()	rdd:DBStruct()
DbUnlock()	rdd:Unlock()
DbUnlockAl()	rdd:Unlock()
DbUseArea()	RddSetDefa(), rdd:Alias[ass], rdd:Close(), rdd:FieldGet(), rdd:FieldPut(), rdd:Used[acc]
Deleted()	rdd:Deleted[acc]
Eof()	rdd:Eof[acc]
Fcount()	rdd:FCount[acc]
Field()	rdd:FieldName()
FieldDeci()	rdd:FieldInfo(DBS_DEC)
FieldGet()	rdd:FieldGet()
FieldGetAr()	DbGoto(), Fcount(), FieldGet(), Recno(), Used()
FieldLen()	rdd:FieldInfo(DBS_LEN)
FieldName()	rdd:FieldName()

FieldPos()	rdd:FieldPos()
FieldPut()	rdd:FieldPut()
FieldPutAr()	DbGoto(), Fcount(), FieldPut(), Recno(), Used()
FieldType()	rdd:FieldInfo(DBS_TYPE)
Flock()	rdd:FLock()
Found()	rdd:Info(DBI_FOUND), rdd:Found[acc]
Header()	rdd:Header[acc]
IdDbFlock()	rdd:Info(DBI_ISFLOCK)
IndexCheck()	rdd:OrderInfo(DBOI_INDEXCHECK)
IndexCount()	rdd:IndexCount
IndexDbf()	rdd:OrderInfo(DBOI_DBFNAME)
IndexExt()	rdd:IndexExt[acc]
IndexKey()	rdd:IndexKey()
IndexNames()	rdd:OrderInfo(DBOI_ORDERCOUNT), rdd:OrderInfo(DBOI_NAME)
IndexOrd()	rdd:IndexOrd()
IsDbExcl()	rdd:Shared[acc]
IsDbRlock()	Recno(), Used(), rdd:RlockList
LastRec()	rdd:LastRec[acc]
LUpdate()	rdd:LUpdate[acc]
MemoExt()	rdd:Info(DBI_MEMOEXT)
NetErr()	
OrdCondSet()	rdd:SetOrderCondition()
OrdCreatte()	rdd:CreateOrder()
OrdDestroy()	rdd:DeleteOrder()
OrdListAdd()	rdd:SetIndex()
OrdListCle()	rdd:ClearIndex()
OrdListReb()	rdd:Reindex()
OrdSetFocu()	rdd:SetOrder()
RddAnnAlia()	
RddAnnFiel()	
RddName()	rdd:Driver[acc]
RddRetAlia()	
RddRetFiel()	
RecCount()	rdd:LastRec[acc]
RecNo()	rdd:RecNo[acc]
RecSize()	rdd:RecSize[acc]
Rlock()	rdd:Unlock(), rdd:Rlock()
Select()	
Used()	rdd:Used[acc]
UsersDbf()	rdd:UsersDbf()

3. Standard RDD drivers

3.1 Available properties of DbfIdx

The standard DBFIDX database driver (RDD) support:

DataServer Name oRdd:	Type	DBFIDX	Note	Dbf/Idx part
Alias	Access,Assign	yes		dbf
AliasSym	Access	no		
Append()	Method	yes		dbf, uses idx
AppendDB()	Method	yes	1	dbf, uses idx
AppendDelimited()	Method	yes		dbf, uses idx
AppendSDF()	Method	yes		dbf, uses idx
AsString()	Method	yes		
Average()	Method	yes		dbf, uses idx
Axit()	Method	yes		dbf, uses idx
BlobDirectExport()	Method	no		
BlobDirectGet()	Method	no		
BlobDirectImport()	Method	no		
BlobDirectPut()	Method	no		
BlobExport()	Method	no		
BlobGet()	Method	no		
BlobImport()	Method	no		
BlobRootGet()	Method	no		
BlobRootLock()	Method	no		
BlobRootPut()	Method	no		
BlobRootUnlock()	Method	no		
BOF	Access	yes		dbf and idx
ClearFilter()	Method	yes		dbf
ClearIndex()	Method	yes		idx
ClearLocate()	Method	yes		dbf
ClearRelation()	Method	yes		dbf
ClearScope()	Method	yes		dbf
Close()	Method	yes		dbf, uses idx
Commit()	Method	yes		dbf
ConcurrencyControl	Access,Assign	yes		dbf
Continue()	Method	yes		dbf, uses idx
CopyDB()	Method	yes		dbf, uses idx
CopyDelimited()	Method	yes		dbf, uses idx
CopySDF()	Method	yes	2	dbf, uses idx
CopyStructure()	Method	yes		dbf
Count()	Method	yes		dbf, uses idx
CreateDB()	Method	yes		dbf
CreateIndex()	Method	yes		idx
CreateOrder()	Method	yes		idx

DataField()	Method	no		
DBStruct()	Method	yes		dbf
Delete()	Method	yes		dbf
DeleteAll()	Method	yes		dbf
Deleted	Access	yes		dbf
DeleteOrder()	Method	no		
Driver	Access	yes		dbf
EOF	Access	yes		dbf, idx
ErrInfo	Access	yes		dbf
Error()	Method	yes		dbf
Eval()	Method	yes		dbf, uses idx
FCount	Access	yes		dbf
FieldGet()	Method	yes		dbf
FieldGetFormatted()	Method	no		
FieldHyperLabel()	Method	no		
FieldInfo()	Method	yes		dbf
FieldName()	Method	yes		dbf
FieldPos()	Method	yes		dbf
FieldPut()	Method	yes		dbf, notifies idx
FieldSpec()	Method	no		
FieldStatus()	Method	no		
FieldSym()	Method	no		
FieldValidate()	Method	no		
FileSpec	Access	no		
Filter	Access,Assign	yes		dbf
FLock()	Method	yes		dbf
ForBlock	Access,Assign	yes		dbf
Found	Access	yes		dbf, idx
GetArray()	Method	yes		dbf, uses idx
GetArrFields()	Method	yes		dbf, uses idx
GetLocate()	Method	yes		dbf
GetLookupTable()	Method	yes		dbf, uses idx
GoBottom()	Method	yes		dbf, uses idx
GoTo()	Method	yes		dbf
GoTop()	Method	yes		dbf, uses idx
Header	Access	yes		dbf
IndexCheck()	Access	yes		dbf, idx
IndexCount	Access	yes		idx
IndexExt	Access	yes		idx
IndexKey	Access	yes		idx
IndexKey()	Method	yes		idx
IndexOrd()	Method	yes		idx
Info()	Method	yes		dbf
Init()	Method	yes		dbf
IsRelation	Access,Assign	yes		dbf
Join()	Method	yes	3	dbf, uses idx
LastRec	Access	yes		dbf

Locate()	Method	yes		dbf, uses idx
LockCurrentRecord()	Method	yes		dbf
LockSelection()	Method	no		dbf
LUpdate	Access	yes		dbf
Name	Access	yes		dbf
NoiVarGet()	Method	yes	4	dbf
NoiVarPut()	Method	yes	4	dbf
NoMethod()	Method	yes	4	dbf
Notify()	Method	no		
OrderBottomScope	Access,Assign	no		
OrderDescend()	Method	no		idx
OrderInfo()	Method	yes		idx
OrderIsUnique()	Method	yes		idx
OrderKeyAdd()	Method	no		idx
OrderKeyCount()	Method	yes		idx
OrderKeyDel()	Method	no		idx
OrderKeyGoTo()	Method	no		idx
OrderKeyNo	Access,Assign	yes/no		idx
OrderKeyNo()	Method	yes		idx
OrderKeyVal	Access	yes		idx
OrderScope()	Method	no		idx
OrderSkipUnique()	Method	no		idx
OrderTopScope	Access,Assign	no		idx
Pack()	Method	yes		dbf, uses idx
QuickFieldGet() *	Method	yes		dbf
QuickFieldPut() *	Method	yes		dbf, notifies idx
RDDInfo()	Method	yes		dbf
RDDName	Access	yes		dbf
ReadOnly	Access	yes		dbf
Recall()	Method	yes		dbf
RecallAll()	Method	yes		dbf
RecCount	Access	yes		dbf
RecNo	Access,Assign	yes		dbf, may notify idx
RecordInfo()	Method	yes		dbf
RecSize	Access	yes		dbf
Refresh()	Method	yes		dbf, notifies idx
RegisterClient()	Method	no		
Reindex()	Method	yes		idx
Relation()	Method	yes		dbf
RelationObject()	Method	yes		dbf
Replace()	Method	yes		dbf, uses idx
ResetNotification()	Method	no		
RLock()	Method	yes		dbf
RLockList	Access	yes		dbf
RLockVerify()	Method	yes		dbf
RollBack()	Method	no		
Scope	Access,Assign	yes		dbf

Seek()	Method	yes	5	dbf, uses idx
SeekEval()	Method	yes	6	dbf, uses idx
SetDataField()	Method	no		
SetFilter()	Method	yes		dbf
SetIndex()	Method	yes		idx
SetOrder()	Method	yes		idx
SetOrderCondition()	Method	yes		idx
SetRelation()	Method	yes		dbf
SetSelectiveRelation()	Method	no		
Shared	Access	yes		dbf
Skip()	Method	yes		dbf, uses idx
Sort()	Method	yes		dbf, uses idx
Status	Access	no		
Sum()	Method	yes		dbf, uses idx
SuspendNotification()	Method	no		
Total()	Method	yes		dbf, uses idx
Unlock()	Method	yes		dbf
Update()	Method	yes	7	dbf, uses idx
Used	Access	yes		dbf
UsersDbf()	Method	yes		dbf
WhileBlock	Access,Assign	yes		dbf
Zap()	Method	yes		dbf, uses idx
IndexAppend()	Prot.Method	yes		see sect. 3.2
IndexGoBottom()	Prot.Method	yes		see sect. 3.2
IndexGoTop()	Prot.Method	yes		see sect. 3.2
IndexReplace()	Prot.Method	yes		see sect. 3.2
IndexSkip()	Prot.Method	yes		see sect. 3.2
IndexSeek()	Prot.Method	yes		see sect. 3.2
IndexSeekCompare()	Prot.Method	yes		see sect. 3.2
IndexSeekEval()	Prot.Method	yes		see sect. 3.2
IndexSynchronize()	Prot.Method	yes		see sect. 3.2
BOF	Prot.Instance	yes		see sect. 3.2
EOF	Prot.Instance	yes		see sect. 3.2
RECNO	Prot.Instance	yes		see sect. 3.2
LASTREC	Prot.Instance	yes		see sect. 3.2
INDEXCOUNT	Prot.Instance	yes		see sect. 3.2
ORDERNUM	Prot.Instance	yes		see sect. 3.2

Notes:

1 oRdd:AppendDB(), expO1 is not supported, use expC1 instead
2 oRdd:CopyDB(), expO1 is not supported, use expC1 instead
3 oRdd:Join(), expO1 and expO2 are not supported, use expC1 and expC2 instead
4 oRdd:NoiVarGet(), NoiVarPut() and NoMethod() are inherited from the DataServer class
5 oRdd:Seek(), expL3 is not supported
6 oRdd:SeekEval(), expL2 is not supported
7 oRdd:Update(), expO1 is not supported, use expC1 instead

3.2 Notes for RDD Programmers

The default DBFIDX driver consist of two independent parts: the database handling and the, on it depending, index part. You may therefore inherit the DbfIdx class into your own class, use the database part and supply/ replace the required index part only.

The index part contains and handles all the exported methods Index*(), Order*(), SetIndex(), SetOrder*(), ClearIndex(), CreateIndex(), CreateOrder(), Reindex() for itself. The database part handles the rest. It communicates with the index part via the protect Bof, Eof, Found, Recno and Lastrec instances, the *Info() methods, as well as via protected methods listed below:

oRdd:IndexAppend () —> retL **Prot.Method**

Notifies the idx part, that a new, empty record has been written to the database file.

Returns: the return value is not used by the dbf part.

Description: The method may now append a new, empty key to all open index files, or wait until the IndexSynchronize() message is sent.

Related: IndexSynchronize()

oRdd:IndexGoTop () —> retL **Prot.Method**

The idx part determines the first logical record number.

Returns: the return value is not used by the dbf part. The method must set the RECNO instance, corresponding to the physical record number, to which the database pointer should be moved. On error, e.g. for an empty database or index, RECNO should be set to LASTREC+1 and and both EOF and BOF to TRUE.

Description: This method is invoked only, if both the INDEXCOUNT and ORDERNUM instances are greater than zero. It has to consider the index scope criteria. Before returning to the application, the dbf part considers all the general scopes and filters, which may result in additional invocations of the IndexSkip() method. If so, no additional IndexSynchronize() messages are sent, since the database is internally locked for the duration of the operation.

Related: IndexGoBottom(), IndexSkip()

oRdd:IndexGoBottom () —> retL **Prot.Method**

The idx part determines the last logical record number.

Returns: the return value is not used by the dbf part. The method must set the RECNO instance, corresponding to the physical record number, to which the database pointer should be moved. On error, e.g. for an empty database or index, RECNO should be set to LASTREC+1 and and both EOF and BOF to TRUE.

Description: This method is invoked only, if both the INDEXCOUNT and ORDERNUM instances are greater than zero. It has to consider the index scope criteria. Before returning to the application, the dbf part considers all the general scopes and filters, which may result in additional invocations of the IndexSkip() method. If so, no additional IndexSynchronize() messages are sent, since the database is internally locked for the duration of the operation.

Related: IndexGoTop(), IndexSkip()

oRdd:IndexSkip (expI1) —> retI *Prot.Method*

Forces the idx part, to skip to the next or previous logical record.

retI = oRdd:IndexNext (expI1)

Arguments: <**expI1**> is the number of records to skip forward if positive, or backward for negative values. Values greater than 1, or smaller than -1 apply only, if no additional scopes of filters are set.

Returns: <**retI**> is the number or records skipped. On success, this value is equivalent to the <expI1> argument. The method must set the RECNO instance, corresponding to the physical record number, to which the database pointer should be moved. On error, e.g. if Eof() or Bof() is reached, the RECNO should be set to LASTREC+1 or to the first logical record respectively, and the EOF or BOF instance correspondingly.

Description: This method is invoked only, if both the INDEXCOUNT and ORDERNUM instances are greater than zero. Before its invocation, the IndexSynchronize(1) message is sent. The IndexSkip() method has to consider the index scope criteria. Before returning to the application, the dbf part considers all the general scopes and filters, which may result in additional invocations of the IndexSkip() method, but without additional IndexSynchronize() messages, since the database is already synchronized and internally locked for the duration of the operation.

Related: IndexGoBottom(), IndexGoTop(), IndexSynchronize()

oRdd:IndexSeek (exp1, expL2) —> retL *Prot.Method*

Seeks the index key for the given value.

retL = oRdd:IndexSeek (exp1, expL2)

Arguments: <**exp1**> is the key value, corresponding to <exp1> parameter of the oRdd:Seek() method.
<**expL2**> is equivalent to the SoftSeek clause. It represents the <expL2> parameter of oRdd:Seek() or the current value of SET(_SET_SOFTSEEK) otherwise.

Returns: <**retL**> signals success, equivalent to FOUND. The method must set the RECNO instance, corresponding to the physical record number, to which the database pointer should be moved. On error, the RECNO and EOF should be set according to the <**expL2**> SoftSeek state.

Description: This method is invoked only, if both the INDEXCOUNT and ORDERNUM instances are greater than zero. Before its invocation, the IndexSynchronize(2) message is sent. The IndexSeek() method has to considers the index scope criteria. Before returning to the application, the dbf part consider all the general scopes and filters, which may result in additional invocations of the IndexSkip() method, but without additional IndexSynchronize() messages, since the database is already synchronized and internally locked for the duration of the operation.

Related: IndexSeekCompare(), IndexSeekEval(), IndexSkip(), Index- Synchronize(), Seek()

oRdd:IndexSeekCompare (exp1) —> retl ***Prot.Method***

Determines whether the current index key is still the SEEKed one.

retl = oRdd:IndexSeekCompare (exp1)

Arguments: <**exp1**> is the value to compare with the current index key. It is equivalent to <exp1> parameter of the oRdd:Seek() or IndexSeek() method.

Returns: <**retl**> signals the result of the comparison: -1: the <exp1> is lower than the current index key 0: the <exp1> is equal to the current index key +1: the <exp1> is greater than the current index key

Description: After a successful IndexSeek(), additional SKIPs may be required when global scopes or filters are set for this working area. If the returned record from IndexSeek() does not fulfill the scope/filter criteria, the dbf part search for the next matching record (if any) via repeated IndexSkip() invocation. To avoid the (relatively slow) macro evaluation of the index key thereafter, the IndexSeekCompare() method is invoked for this comparison. The IndexSynchronize() message is not sent, since the database is already synchronized and internally locked for the duration of the operation.

Related: IndexSeek(), IndexSkip()

oRdd:IndexSeekEval (expB1, expL2) —> retL ***Prot.Method***

Skips through the index, starting at the current position, searching for the next key for which the given code block returns TRUE.

retL = oRdd:IndexSeekEval (expB1, expL2)

Arguments: <**expB1**> is the code block, corresponding to the <expB1> parameter of the oRdd:SeekEval() method.
<**expL2**> is equivalent to the <expL2> parameter of oRdd:SeekEval() or TRUE if not given there.

Returns: <**retL**> signals success, equivalent to FOUND. The method must set the RECNO instance, corresponding to the physical record number, to which the database pointer should be moved. On error, the RECNO and EOF should be set accordingly.

Description: This method is invoked only, if both the INDEXCOUNT and ORDERNUM instances are greater than zero. Before its invocation, the IndexSynchronize(3) message is sent. The IndexSeekEval() method has to consider the index scope criteria. When <expL2> is set to TRUE, the oRdd:GoTo(Recno) method has to be invoked and the RECNO instance updated for every index key movement. Before returning to the application, the dbf part considers all the general scopes and filters, which may result in additional invocations of the IndexSkip() method, but without additional IndexSynchronize() messages, since the database is already synchronized and internally locked for the duration of the operation.

Related: IndexSeek(), IndexSkip(), IndexSynchronize(), SeekEval()

oRdd:IndexReplace () —> NIL *Prot.Method*

Notifies the idx part, that the current record has been changed and written to the database file. The method should now check and update the index keys of all open index files/orders.

Description: Before its invocation, the IndexSynchronize(4) message is sent, whereby the fields correspond to the original database values, before replacement. At the time of the IndexReplace() invocation, the current field contents (available via the oRdd:FieldGet() method or by accessing the field name) corresponds to the new state of the database. The IndexReplace() method has to consider the index scope criteria, such are conditional index and the descend flag.

Multiuser hint: you may store the original index key values when IndexSynchronize(4) is received, to compare them with the newly evaluated values here. If the key value remains unchanged, the index file/order does not need to be changed.

Related: IndexSynchronize(), IndexAppend()

oRdd:IndexSynchronize (expl1) —> NIL *Prot.Method*

Notifies the idx part of the RDD, that an index pointer synchronization is required for a subsequent index movement or key replacement. Designed to support optimized index access.

oRdd:IndexSynchronize (expI1)

Arguments: <**expl1**> announces a forthcoming index action

1: IndexSkip() follows, must synchronize.Invoked only if INDEXCOUNT > 0 and the ORDERNUM instance is not 0.
2: IndexSeek() follows, may synchronize. Invoked only if INDEXCOUNT > 0 and the OrderNum instance is not 0.
3: IndexSeekEval() follows, must synchronize. Invoked only if INDEXCOUNT > 0 and the ORDERNUM instance is not 0.
4: IndexReplace() follows, may/must synchronize. Invoked if INDEXCOUNT > 0, regardless of the ORDERNUM value.

Description: It is not required, that the index pointer always corresponds to the database pointer, especially when SET ORDER is set TO 0 and the database is skipped in natural record order, or when several GOTO statements are performed. The synchronization is required before an index movement request at latest, or before an index key replacement. The IndexSynchronize() message notifies the idx part, that the current field values exactly reflect to the database record contents.

The body of the IndexSynchronize() method may determine the new index values (e.g. by evaluating the IndexKey() string) for all open indices, and perform an internal seek for this value/ record in the index file(s) to synchronize the index pointer, if required.

Where necessary in your idx methods, you may force the synchronization, including the replacement of pending changes, via oRdd:Skip(0) or oRdd:Commit(). The latter will not send any IndexSynchronize() message, except if a replacement is pending.

Related: IndexReplace(), IndexAppend(), IndexSkip()

oRdd:Bof <–> expL *Prot.Instance*

Begin-of-file flag, set by the dbf part of the RDD or the Index- GoTop(), IndexGoBottom(), IndexSkip() methods.

oRdd:Eof <–> expL *Prot.Instance*

End-of-file flag, set by the dbf part of the RDD or the IndexGoTop(), IndexGoBottom(), IndexSkip(), IndexSeek(), IndexSeekEval() methods.

oRdd:Recno <–> expI *Prot.Instance*

Current physical record number (1...LASTREC+1), set by the dbf part of the RDD or the IndexGoTop(), IndexGoBottom(), IndexSkip(), IndexSeek(), IndexSeekEval() methods.

oRdd:LastRec <–> expI *Prot.Instance*

The last valid physical record number (greater than or equal to 0), set by the dbf part of the RDD.

oRdd:IndexCount <–> expI *Prot.Instance*

Number of open indices (0..15) being used in this RDD. Managed by the idx part of the RDD in oRdd:Ord*() and used also e.g. in the IndexCount() function.

oRdd:OrderNum <–> expI *Prot.Instance*

Ordinal number of the currently controlling index (0..15) in the list of open indices for this RDD. Managed by the idx part of the RDD in oRdd:Ord*() and used also e.g. in the IndexOrd() function.

4. Third Party RDDs

If you own additional 3rd party RDDs, you may insert the description here.

Index

multisoft Datentechnik
Schönaustr. 7
D-84036 Landshut

http://www.fship.com
sales@multisoft.de
support@flagship.de