

DISAM References and History

- [DISAM History, Version 7](#)
- [DISAM History, complete](#)
- [upgrading from old disam](#)
- [upgrading from cisam](#)
- [base library reference](#)
- [answers to frequently asked questions](#)
- [suggestions, pitfalls and defenses](#)
- [audit trail mechanism](#)
- [large files – 64bit implementation](#)
- [custom key types](#)
- [design concepts](#)
- [disam client/server reference](#)
- [dcheck](#)
- [dpack](#)
- [dreport \(new\)](#)
- [dschema \(new\)](#)
- [dsplit](#)
- [dvlreblid \(new\)](#)
- [dstats](#)
- [expiration date](#)
- [isconfig.h ISEXTENDED configuration setting](#)
- [isread filtering](#)
- [user info](#)
- [installation](#)
- [key compression](#)
- [transparent file links](#)
- [index masking](#)
- [isam file mirroring reference](#)
- [null key masking](#)
- [pure indexes](#)
- [auto file corruption repair](#)
- [DISAMv7.0 schema manipulation routines](#)
- [status variables](#)
- [bench test source](#)
- [threadsafe](#)
- [transaction processing](#)
- [variable length records](#)
- [wrapper reference](#)

DISAM History, Version 7

jan, 2009 – official release of DISAM v7.0 new to this release

isopen.c, isbuild.c, isvarlen.c, isvarlen.h and isconfig.h – changes to variable length idx file format to allow for > 16777216 record limit when storing record numbers in 3 char bytes for varinfo and continuation node pointers. isconfig.h setting ISOLDVARLEN 1 or any non 0 ISCISAM setting will maintain old disam/c–isam variable length file format compatibility. ISCISAM 0 and ISOLDVARLEN 0 ensures new format. new format requires that dvlreblid(see dvlreblid.ref) be run on existing variable length files.

stdextra.c – schema/record layout support – new wrap functions
load schema from isamfile to path/filename
ISD1 int ISD2 isldschem(int isfd, char *path)

Ref Files/FAQ's

dump schema from isamfile to stdout
ISD1 int ISD2 isdpschema(int isfd)
store schema from loadfile to isamfile
ISD1 int ISD2 isstschema(int isfd, char *loadfile)
remove schema from isamfile
ISD1 int ISD2 isrmschema(int isfd)

isschema.c – schema record layout support – new base functions

load schema from isamfile to path/filename
ISD1 int ISD2 isLdSchema(IsFile *isam, char *path)
dump schema from isamfile to stdout
ISD1 int ISD2 isDpSchema(IsFile *isam)
store schema from loadfile to isamfile
ISD1 int ISD2 isStSchema(IsFile *isam, char *loadfile)
remove schema from isamfile
ISD1 int ISD2 isRmSchema(IsFile *isam)

* see read/schema.ref for function details and schema/ReadMe for sample programs

dreport.c – new schema related isam data reporting utility.

record number limit changed from LONG_MAX to ULONG_MAX value

client/server capabilities incorporated into standard DISAM. for a free limited single client server program please contact ByteDesigns.

DISAM History, complete

The first DISAM was released in March of 1988.
This history commences with Version 5.

september 24th, 1995 – version 5.93 – first official release

october 3rd, 1995 – version 5.94

this version incorporates the initial release of variable length data records, and two minor bug patches relating to data record locks and natural order read/search.

october 8th, 1995 – version 5.95

this version includes an optional administration facility to keep track of active isam file descriptors.

the administration facility allows the library to keep track of duplicate opens on isam files and to share a single pair of system file handles between duplicate instances of the same file.

ISMAXFILES and ISMAXLK are now redundant, and the associated tables are now dynamically allocated, meaning that the number of open files

Ref Files/FAQ's

and the number of active record locks is now only limited by system resources.

the initial beta version of the transaction processing module is included. this version features an enhancement over the original disam code in that it supports virtual open/close within transactions. any attempt to close a file that is part of an active transaction will be deferred until the transaction is complete. if you chose to reopen the file within the same transaction, the virtual handling system will just pass you back the same handle you were using before. this, in theory, should help to correct the EMFILE errors inherent in the original disam versions. refer to trans.ref for details.

the test/user.c executable has been enhanced slightly in that it can now make system calls (sys) and run rudimentary scripts (run).

october 16th, 1995 - version 5.97

featuring a number of adjustments and enhancements to the variable length option, and the implementation of a fairly rigorous storage checking/statistics facility in ascheck.

the remainder of the changes cover basic adjustments for portability and a degree of cleanup and optimisation in the remainder of the code.

october 31st, 1995 - version 5.98

initial beta - threadsafe and threadproof handling - see thread.ref for more details.

implemented true support for duplicate opens within the same process. record locking on duplicate handles is honoured, and locks will not be lost when a duplicate handle is closed.

isrecover now returns appropriate error codes on failure.

december 15th, 1995 - version 5.99

essentially a collection of adjustments based on beta test reports. added isprecious() and islockcheck() options to the wrapper code.

january 2nd, 1996 - version 6.0

the updates in this version consist of the inclusion of the isCluster and isAudit calls, documented in base.ref and audit.ref respectively.

january 28th, 1996 - version 6.1

the transaction processing module has been upgraded to comply with the current cisam standard, and includes full support for variable length records. disk i/o has been improved (at the expense of an extra chunk of memory) and the lock time on the file log reduced.

added ISDUPLOCKS option to the configuration options to permit cisam (non xopen) functionality with regard to duplicate opens in the same process – see entry for version 5.98 for more detail. please note that ISDUPLOCKS should not be set to zero when porting to DOS, Windows, OS/2 or non ELF versions of Linux. if this is a problem to you, please contact Byte Designs.

the flag structure in the index header that denotes ISMASKED files has been changed. this means that masked files built before this release are no longer compatible. the fixflags utility in the util directory will patch existing files to comply with the new method.

added isindex, ispop, iscopy and isclone calls – see upgrade.ref for further details.

the ascheck utility now includes a -B option to allow specific indexes to be rebuilt without disturbing the remainder. see ascheck.ref for more details.

february 5th, 1996 – version 6.2

isClose now returns a boolean TRUE or FALSE rather than the original NULL pointer. this change is specific to the base library and does affect the standard interface.

base and wrapper function prototypes are now C++ compatible.

added isseekey() to the wrapper library.

ascheck now does index/data mismatch checking – slower but better.

aslist in the util directory is an incomplete but fully functional means of listing the contents of an isam index file.

added isPush and isPop to the base library.

added threadsafe isvbuild to the wrapper to allow building variable length files without the use of the global isreclen variable.

added isWrCurr and iswrcurr to the base and wrap libraries.

added isfilter (wrapper only) and islcall (inhouse only)

pure index files – see pure.ref

transparent file links – see link.ref

Ref Files/FAQ's

stubs for isstat variables – see status.ref

added ISRONLY mode in isopen for handling read only files.

february 14th, 1996 – version 6.4

null key masking – see nullkeys.ref

custom key extensions – custkeys.ref

RLE compression option for variable length – ISVARIABLE in install.ref

renamed isLockMode and islockmode to isSetMode and issetmode.

implemented ISSYNCWR mode in open and build – see base.ref

upgraded isprecious logic to complete implementation

adjusted isGoto to not return an error if file not open for reading.

version 6.5 was an interim release, changes from 6.4 covered below.

february 14th, 1996 – version 6.6

threadsafe transaction processing module – beta only – see trans.ref for details.

added ISVARCMP mode on isbuild to allow runtime compression flagging on variable length files, adjusted variable length compression handling, and added ISNOCARE option to isOpen to allow easier handling of varlen files – see varlen.ref for details.

reworked threadsafe duplicate open handling.

added EINTR handling to lockmode 2 for threadsafe, and allowed multiple lock fail codes to be specified in isport.h

stchar arguments were reversed, and the float machine independant routines now have float arguments rather than double.

open and build now allow the lock mode to default to ISMANULOCK if not specified.

adjusted isStart behaviour to match cisam standards.

cleaned up a number of compiler warnings.

Ref Files/FAQ's

march 22nd, 1996 – version 6.7

allow varlen compression to be specified on a per file basis,
see varlen.ref

implemented ISNOCARE open mode to make it easier to open unknown
(varlen or fixlen) files. also modified isgetmode to get all
open mode bits rather than masking out only the lock mode, and
adjusted issetmode to accept an unmasked mode value without
complaint. see varlen.ref

added common dictionary access – see wrap.ref, should be of
value to those converting from old disam.

error code offset value define – see install.ref

added varlen and varcmp mention in ascheck.

display contents of known key types in aslist.

user info pad in index headers – see info.ref

april 10th, 1996 – version 6.8

adjusted ISDUPLOCKS 1 mode to disallow erase/rename on files already
open in the current process – DUPLOCKS 0 retains original behaviour,
but be warned that a sequence of open, erase, write, close will result
in data loss – see install.ref and caveat.ref for details.

added isdi_kdsc() to the wrapper library – see wrap.ref

library expiration date option – see expire.ref

may 3rd, 1996 – version 6.9

added NPARTS define for cisam/xopen compatibility.

completed ISDATAVOID conditioning in the wrapper interface.

added missing TNULL key compression option.

corrected STRINGTYPE key loading bug.

added missing key description validity checks.

no longer legal to erase the last remaining index in a file.

overflow on opening files with too many indexes no longer fatal.

isdelcurr will now fail after isstart first/last.

Ref Files/FAQ's

adjusted error return code in isrewcurr and isdelcurr on empty file.

corrected dislocation problems caused by use of is_recnum().

revised error return conditioning in isread and isstart to ensure appropriate error returns under various conditions.

isread ISNEXT now returns current after ALL isstart operations except isstart ISNEXT.

corrected non-portable bug when selecting descending order indexes.

adjusted isgoto to handle natural order index properly.

isolated natural order current.

null flag in stfltnull and stdblnull should not have been a pointer.

aspack was not handling null primary indexes properly.

august 18th, 1996 - version 6.10

much pruning and transplanting..

the source is now distributed in two flavours - standard unix format, and a dos/win/os2 version. both are generated from the same meta source but contain unique platform extensions.

base/ismemory.c and wrap/stdmif.c must be added to make/project.

now only four headers. isconfig.h is common to all, isbase.h and iswrap.h as before, iscustom.h is optional.

isport.h is gone, as is ISPLATFORM. BSD extensions and threadsafe configuration are now in isconfig.h

added isLockCheck to the base interface, parallel to islockcheck in the wrapper.

the base now has it's own mif calls - IdInt, stDbINull, etc, and these have been wrapped for access from the standard interface.

added isErrno, isErrio, isRecNum, isRecLen, isSetRec, isSetLen calls to base.

additional support for DLL compilation included, see dynamic.ref for more details.

additional support for microsoft windows added, see windows.ref for more details.

misc/dbmemory.c is a debugging replacement for ismemory.c, see same for comments and usage.

Ref Files/FAQ's

added ISLOCKING 3 for dos based systems and support for 'lowest common denominator' NFS based configurations.

the built in check call prototypes, codes and structures are now included in both the base and wrapper headers, and ischeck.h is no longer provided or required.

entry function prototyping has been revised to provide more flexible support for building dynamic link libraries.

the general lock handling scheme has been revised somewhat to cover requirements arising from the ISLOCKING 3 option.

corrected a relatively obscure memory under-run bug in the tree pruning module.

added isLastRec(), isGetLastRec(), isSetLastRec() to the base, islastrec(), isgetlastrec(), issetlastrec() to the wrapper. must add base/islast.c and wrap/stdlast.c respectively to make/project to use. undocumented (as yet)

added base library file descriptor sanity checks

corrected invalid eof on read next after adding records to an empty file.

pure indexes not following isfirst/isnext rules after delete and erase corrected.

transaction processing – commit/rollback was failing when no updates made within transaction.

isStart/isstart should not allow the user to set keylength greater than the length of the selected key.

isCopy/iscopy was reporting a bogus error on success.

build and addindex were being a bit too fussy about the contents of a null key descriptor.

isindexinfo and related were not counting null primaries.

lock handling not releasing internal record lock state flags under some situations.

october 16th, 1996 – Version 6.11 – James Knight bids his official farewell

Byte Designs assumes ownership of Disam96 – changes to copyright/banners

ascheck, aspack and aslist references changed to dcheck, dpack and dlist respectively

sept 1st, 1997 – Version 6.12

Ref Files/FAQ's

dec modules standardize to fit disam96 function prototyping

isclone ensures null primary is cloned

compnext now returns FALSE if no current record

isopen reuses system fds on failure

dpack copies unique ids on rebuild

dcheck added -q quiet mode option

iserrno initialization on successful return

int64 support for maintenance of large files, ISHUGE 1 in isconfig.h

cisam version 7.1 locking concurrency added, C7LOCKING 1 in isconfig.h

rewnxt/RewNxt now correctly returns EENDFILE rather than looping on last record

extensive transaction processing rewrites

cisam 7.1 no-wait on open and automatic record locks, C7NOWAIT 1 in isconfig.h

cisam isstat cobol variables implemented

may 1st, 1999 - Version 6.13 - Client / Server

client/server capability - see client.ref

file mirroring capability - see mirror.ref

january 15th, 2000

additional c-isam behaviour implementations - C7NOWAIT set to 1
ISSKIPLock and no skip on locked records

fix to ensure deletions of compressed index values don't cause uncompression
of previous index values to bump past end of node - caused 105 EBADFILE errors

may 1st, 2000

c-isam version 7.2x locking concurrency support

isconfig.h - C7LOCKING setting should be set to 2

july 5th, 2000

Ref Files/FAQ's

added the utility – dsplit – allows a large file to be split into two smaller files.

january 16th, 2001

added libinfo structure in association with new subroutines isLibInfo() and islibinfo(). isLibInfo()/islibinfo() when called with an empty libinfo pointer, will load the libinfo structure with the expiration date, if set, the library compilation date and the number of clients if set.

added asplog flag to ensure that socket error reporting within the client code, is logged to a file rather than displayed. new functions, AspGet() and AspSet(int flag) in the base library code and, aspget() and aspset(int flag), will set and report the current value of asplog.

several refinements to the client/server code..extensive log reporting.

added dstat utility that reports library statistics, see dstats.ref for more information.

may 1st, 2001

added auto repair for index/data corruptions that may occur if a process is killed or if a system failure occurs. see repair.ref and ISREPAIR setting in isconfig.h

september, 2001

added ISCISAM 2 option – same as ISCISAM 1, ie. mimic C-ISAM behaviours, but with the addition of ENOCURR on isread/ISEQUAL failure.

duplicate count logic – rev.2 – isgrow.c effected.

added check for ISWAIT on AUTOLOCK open with ISCISAM > 0 or ISNOWAIT 1

november, 2001

isIndexInfo/isindexinfo – EBADARG(102) when file opened with other than ISINOUT
versions effected: 6.12 unix and MS, 6.13CS unix and MS – isinfo.c

isupdate.c, isdelete.c – increased allocated size of oldpad buffer
general reorganization of rewrite() and delete() routines for better efficiency.c
versions effected: 6.12 unix and MS, 6.13CS unix and MS

added more functionality to ISAUDIT – additional options 2 through 7 see audit.ref
tune which operations are audited, see audit.ref for options
versions effected: 6.12 unix and MS, 6.13CS unix and MS
isconfig.h, isdelete.c, isupdate.c and iswrite.c

Ref Files/FAQ's

january, 2002

addition of remote system functions – eg. fread, fwrite etc.

may, 2002

ISKCOMP – compile time flag – isconfig.h – see compkey.ref for details

isprune.c replacement to correct index count and fatal errors encountered during continuous rewrite cycles where the key values are effected

aug, 2002

isadmin.c – correction to reference ADM(id)->isam->openmode should read ADM(did)->isam->openmode

isrepair.c – modifications to correct repaired writes

isadmin.c, iserase.c – iserase() failure caused admin table corruption

isdelete.c – drop record lock prior to adding record number to free list

sep, 2002

isprune.c – corrected bug that caused index mismatch errors under certain conditions

ISLOCKING 3 on unix platforms would incorrectly initialize isam->iserrno to 0, overwriting a valid error value

isadmin.c – dup struct pointer not freed on open failure, corrected

nov, 2002

isprune.c – dupnum calculation on insertion into new root corrected

jan, 2003

isbuild.c, isopen.c, isopen.c and isbase.h – added CISAM compatible ISSEMILOCK. see #define SEMILOCK in isconfig.h

feb, 2003

issystem.c, isconfig.h, iswrap.h, isbase.h, stdwrap.c and stdwrap.h added ISEXTENDED compile time option to allow for extended lock reporting when locks are blocked by other processes. see read/extended.ref

Ref Files/FAQ's

mar, 2003

isIndexInfo/isindexinfo - EBADARG(102) when file opened with other than ISINOUT
versions effected: 6.12 unix and MS, 6.13CS unix and MS - isinfo.c

may, 2003

ISCISAM > 0 check for natural order - isread.c
ISCISAM > 0 allow ISOUTPUT record locking, ie. isLock()/islock()

aug, 2003

isconfig.h - D3XAUTOLOCK set to 1 in isconfig.h prior to compiling the library will enable older D-ISAM v3.x behaviour releasing record locks on subsequent read, update or delete

ischeck.c - additional check for last datfree corruption

jan, 2004

isgrow.c - change to satisfy C-ISAM bcheck higher level node ordering problem did not effect the integrity of the isam file or Disam96's ability to maintain the file.

feb, 2004

isread.c - change to ensure partial key searches are C-ISAM compatible when ISCISAM set to non-zero
Note difference between C-ISAM and Disam96 in the following scenario.
Set key compare length to partial key length via isstart()
Given 3 records with key values of 1110, 1120 and 1130
setting compare length to 1 will result in
isread/ISFIRST - first record with key 1110 is returned
isread/ISGREAT - iserrno 111 - ENOREC - Disam96
Since only the first byte of the key is compared and none are greater than the value of 1. With C-ISAM the isread/ISGREAT will return the key 1120 record, which means the isstart setting of partial key length compare to 1 doesn't have any effect.

mar, 2004

isgrow.c - previous change to satisfy C-ISAM bcheck not quite correct for uncompressed keys - refined/corrected. compressed key logic correct.

jul, 2004

Ref Files/FAQ's

isadmin.c – rewrite of dup mutex handling on failure. Mutexes were being unlocked rather than destroyed on function failure, isAdmFail

isentry.c – reversal of isAdmFail() and isDropLock() in isFail(). mutexes were being destroyed before being unlocked. This problem first appeared on Itanium where stricter control of mutexes exist. All platforms effected.

isread.c – ISCISAM 2 specific – ensure check for ISPATH->recnum is only done when reading by index and not for natural or record number order reads.

sep, 2004

iserase.c – Correction of bug that allowed erase/rename of locked file.

nov, 2004

isread.c – change in ISCISAM 2 ENOCURR check

aug, 2005

isadmin.c, stdextra.c, isbase.h – changes to correct isAdmFail master lock end rather than drop. addition of new function isThreadedDone()/ isthreadeddone() to drop master lock. application code should call isthreadeddone() prior to the application exiting..closing to isthreaded() call.

oct, 2005

isaudit.c – changes to correct behaviour to conform to C-ISAM/X-OPEN, original behaviour can be reinstated by defining D96ORIG in isaudit.c

feb, 2006

islock.c – correction to only set bit in lkstate for lock all and not record

sep, 2006

isprune.c – reset node->recnum after compressed key delete

nov, 2007

isbase.h, iswrap.h, isread.c – isStart() added C-ISAM ISKEEPLOCK functionality
isopen.c – ensure isAdmClose if isopen fails while logging in progress

isvarlen.c – allocation of isam->hashtab changed from hardcoded 4 to

Ref Files/FAQ's

sizeof(long *), to ensure proper support of platforms where long is > 4 bytes, namely native 64bit environments.

jan, 2008

istrans.c – replaced missing call to isVlread in log_vdel routine to ensure isrollback() reinserts variable length portion of deleted record.

jan, 2009 – official release of DISAM v7.0 new to this release

isopen.c, isbuild.c, isvarlen.c, isvarlen.h and isconfig.h – changes to variable length idx file format to allow for > 16777216 record limit when storing record numbers in 3 char bytes for varinfo and continuation node pointers. isconfig.h setting ISOLDVARLEN 1 or any non 0 ISCISAM setting will maintain old disam/c–isam variable length file format compatibility. ISCISAM 0 and ISOLDVARLEN 0 ensures new format. new format requires that dvlreblc(see dvlreblc.ref) be run on existing variable length files.

stdextra.c – schema/record layout support – new wrap functions

load schema from isamfile to path/filename
ISD1 int ISD2 isLdschema(int isfd, char *path)
dump schema from isamfile to stdout
ISD1 int ISD2 isDpschema(int isfd)
store schema from loadfile to isamfile
ISD1 int ISD2 isStschema(int isfd, char *loadfile)
remove schema from isamfile
ISD1 int ISD2 isRmschema(int isfd)

isschema.c – schema record layout support – new base functions

load schema from isamfile to path/filename
ISD1 int ISD2 isLdSchema(IsFile *isam, char *path)
dump schema from isamfile to stdout
ISD1 int ISD2 isDpSchema(IsFile *isam)
store schema from loadfile to isamfile
ISD1 int ISD2 isStSchema(IsFile *isam, char *loadfile)
remove schema from isamfile
ISD1 int ISD2 isRmSchema(IsFile *isam)
* see read/schema.ref for function details and schema/ReadMe for sample programs

dreport.c – new schema related isam data reporting utility.

record number limit changed from LONG_MAX to ULONG_MAX value

client/server capabilities incorporated into standard DISAM. for a limited single client server program please contact ByteDesigns.

upgrading from old disam

upgrading from old disam

the process of upgrading from older versions of disam should be as simple as relinking with the new library, but there are a few things

Ref Files/FAQ's

that are different, and may need addressing.

the header file

the standard header file as distributed is named iswrap.h to avoid confusion with older versions and to differentiate from the alternate interface header, which is called isbase.h

the simplest procedure is to replace the contents of your original disam.h with a copy of iswrap.h

alternatively you might wish to create a new copy of disam.h which does nothing more than include either iswrap.h or a renamed copy of your old disam (or cisam) header file. this would permit you to easily switch back and forth until you are comfortable with the new library.

the trouble with isfdmap

the old disam library used an array of pointers, called isfdmap, to manage it's table of open files. the new library uses an internally manipulated dynamic array that is not quite as easy to get to, and the associated file descriptor structure has changed significantly.

analogs for the more common values are now available via the common dictionary access routines described in wrap.ref

please let support know if anything important has been left out.

virtual file open handling (ISVIRTUAL define in old isport.h)

older versions of disam included a compile time option for virtual file handling support. this was activated by means of the ISVIRTUAL define in isport.h

it was designed to allow the library to bypass the system file handle limit by swapping out (closing) inactive system handles to make room for new opens.

because it is not possible to retain locks on closed system handles, this feature introduced a bug that created problems with index access concurrency, sometimes resulting in index corruption.

at time of writing there appears to be no practical solution to this problem, and virtual file handling support has been discontinued in the current release.

Ref Files/FAQ's

the current library provides an alternative solution (although not as extensive) via duplicate open handling. this feature, active by default, allows the library to multiplex the system handles used to access multiple opens on the same isam file.

in other words – you can open the same isam file as often as you wish without the need for additional system handles.

while this does not cover instances where the application needs to access large numbers of different isam files, it should be of value under the circumstances denoted.

no specific configuration is needed to access this feature, as it is an integral part of standard operation.

```
isindex( int isfd, int idx, int locate );
```

the idx argument has been changed. under old disam, a value of zero was used to select the primary index, and a value equal to the max number of active indexes was used to select natural order.

the new library uses a value of zero to denote natural order, and the active index count starts at one.

if you prefer the original functionality, please refer to the pre-processor directive within the islocate code in stdread.c

```
ispop( int isfd, int idx, long rec );
```

the idx argument has been changed. under old disam, a value of zero was used to select the primary index, and a value equal to the max number of active indexes was used to select natural order.

the new library uses a value of zero to denote natural order, and the active index count starts at one.

if you prefer the original functionality, please refer to the pre-processor directive within the ispop code in stdread.c

if you change ispop functionality you should make the same change in ispush.

```
ispush( int isfd, int *idx, long *rec );
```

since the new library no longer supports the isfdmap array, this call has been added to load the current index and record numbers for later

Ref Files/FAQ's
use by ispop.

```
isclone( int isfd, char *name );
```

you can no longer specify the mode of the new file, nor list the indexes to be copied. if this is a problem to you please let the support department know – the option was dropped because it was felt that it had limited use and the associated varargs processing caused an amount of difficulty in porting.

the current version copies all indexes by default, and the open mode of the returned handle will be the same as for the original file.

```
iscopy( int isfd, char *name );
```

iscopy no longer requires that you pass a buffer to assist in the copying process, and the extraneous flags argument has been dropped because it was never used.

```
isgetmode( int isfd, int *mode );
```

this routine loads the open mode value into the mode pointer passed

ISAUTOLOCK behaviour

Disam by default release automatic record locks only on a subsequent read. For older D-ISAM v3.x behaviour set isconfig.h D3XAUTOLOCK to 1 to enable automatic release on read, update or delete

Worldwide Copyright (c) Byte Designs Ltd (2009)

upgrading from cisam

upgrading from cisam

the process of upgrading from cisam should be as simple as relinking with the new library, but there are a few things that you should know about.

the header file

isam.h is cisam's standard header file, ours is called iswrap.h

there are three basic alternatives:

1. modify your application code to include iswrap.h
 2. replace the contents of isam.h with a copy of iswrap.h
 3. create a new copy of isam.h which does nothing more than include either iswrap.h or a renamed copy of isam.h. this would permit you to easily switch back and forth until you are comfortable with the new library.
-

what happens when you change the current key on rewrites

the best way to explain this is by example. assume you have a file containing the following records: AAA, BBB, CCC, EEE, and that the current index is based on these values.

under cisam, if AAA was the current record and you were to change it's contents to DDD and rewrite, an isread or isstart with a mode of ISNEXT would set the current to EEE.

this can be somewhat problematic if you are doing a read next loop through your file, so we have arbitrarily adjusted this behaviour to ensure that the next record found will follow the current as set BEFORE the rewrite.

this means that, given the above example, BBB would become current on an isread or isstart with a mode of ISNEXT.

the same (in reverse) is true when reading backwards through the file with a mode of ISPREV.

we currently consider this behaviour to be an enhancement over the cisam standard, but would be happy to provide true cisam operation if anyone should ask for it. please contact the support department if this is important to you.

cisam autolock / nowait

for cisam compatible nowait on open and automatic record locks set C7NOWAIT to 1 in isconfig.h

cisam version 5.0 through 7.1 locking concurrency

for cisam version 5.0 through 7.1 locking concurrency when running

Ref Files/FAQ's

against isam executables set C7LOCKING to 1 in isconfig.h.
keep in mind that executables compiled with C7LOCKING set to 1
will no longer run concurrent with executables compiled for standard
isam locking.

isam version 7.2x locking concurrency

same as above, except C7LOCKING should be set to 2

Worldwide Copyright (c) Byte Designs Ltd (2009)

base library reference

base library reference

the library provides two separate interface methods. the base code
uses allocated file descriptors (similar to fopen/fwrite etc) to
manage isam files, and it will be this interface that is described
here. the differences between the base and the wrapper interfaces
is covered in wrap.ref

primary details

all modules which make base library calls should include isbase.h

in all of the following, isfd will be used to refer to an open isam
file descriptor, as returned by isOpen or isBuild, and declared as:

```
struct IsamFile *isfd; /* the formal way */  
IsFile *isfd; /* typedef shortcut */
```

return values and result indicators

with the exception of isBuild, isOpen and a few others, the base will
return a boolean ISTRUE or ISFALSE. ISTRUE and ISFALSE are defined
in the isbase.h header as 1 and 0 respectively.

when ISFALSE is returned you will find the system or isam error code
in isfd->iserrno, where isfd is an open file descriptor. isam error
codes start at 100 (refer to ISERRBASE in install.ref if you need to
change this) and are listed under error codes below. system error
codes have their usual values for the target system.

if iserrno is less than 100 (ie a system error) isfd->iserrio will

Ref Files/FAQ's

provide a little more detail as to where and why, as listed under error codes below.

in situations where no file descriptor is available, such as when isBuild or isOpen fail, or after isClose, isErase or isRename, the call will return the error code in the global errno variable.

```
IsFile *isBuild( char *name, int dlen, int mlen, IsKdsc *key, int mode );
```

name: [path]name of file to be created
dlen: length of data record in bytes
mlen: if ISVARLEN then max record length else ignored
key: primary index description, see isam indexes
mode: see open modes
return: allocated file descriptor or NULL

note: if successful, isBuild will return an isam file descriptor open in the specified mode and ready for use.

a NULL pointer is returned on failure and the error code will be found in the global errno variable.

refer to varlen.ref for more details regarding ISVARLEN

```
IsFile *isOpen( char *name, int mode );
```

name: [path]name of an existing isam file
mode: see open modes
return: allocated file descriptor or NULL

note: the file name must be specified without the .dat or .idx extension, and must be less than 251 bytes in length.

the current index will be set to the primary.

a NULL pointer is returned on failure and the error code will be found in the global errno variable.

```
int isCleanUp( void )
```

note: this routine will close all currently open isam files and release all allocated global memory.

only available if the base library was compiled with the ISADMIN feature activated.

```
IsFile *isLockCheck( IsFile *isfd, int mode );
```

isfd: isam file descriptor
flag: 0: off 1:on

note: a flag value of 1 will cause the library to fail on update or delete if the record in question has not been successfully locked prior to the operation. the iserrno value returned in this instance is ENOCURR.

```
IsFile *isSetMode( IsFile *isfd, int mode );
```

isfd: isam file descriptor
mode: either ISMANULOCK or ISAUTOLOCK

note: this routine can be used to switch the designated file between manual and automatic locking modes.

when switching from automatic to manual, the current record, if locked, will be released.

future plans include the ability to switch to/from exclusive lock as well, but at present an attempt to do so will result in an EBADARG error.

```
IsFile *isClose( IsFile *isfd );
```

isfd: isam file descriptor

note: isClose drops all locks and releases all allocated memory.

isClose will return FALSE on failure and the error code will be found in the global errno variable.

```
int isAddIndex( IsFile *isfd, IsKdsc *key );
```

isfd: isam file descriptor
key: description of secondary index to add, see isam indexes

note: when adding an index to a populated file, isAddIndex will load the new index with keys for all the existing data records. in the case of larger files this could take a while..

isAddIndex will attempt to place an exclusive lock on the isam file for the duration of the update (assuming you have not done so already) and will fail if any other

Ref Files/FAQ's

process already has the file open.

```
int isDelIndex( IsFile *isfd, IsKdsc *key );
```

isfd: isam file descriptor

key: description of secondary index to delete, see isam indexes

note: when deleting an index from a populated file, isDelIndex will purge the accumulated keys. in the case of larger files this could take a while, but will be significantly faster than the isAddIndex call.

isDelIndex will attempt to place an exclusive lock on the isam file for the duration of the update (assuming you have not done so already) and will fail if any other process already has the file open.

```
IsFile *isCluster( IsFile *isfd, IsKdsc *key );
```

isfd: isam file descriptor

key: description of index to cluster on

note: the file must be open in exclusive mode

this routine makes use of a temporary file, created in the same directory as the original.

all active data records will be transferred to the temp file in the index order specified, and all indexes will be packed as efficiently as possible.

if successful, isCluster will return a new file descriptor, opened in the same format as the original, which will be abandoned. a NULL pointer will be returned on error, and the original descriptor and file will be left intact.

be warned that this call may take some time to run if the file in question is large.

```
int isErase( char *name );
```

name: [path]name of an existing isam file to be physically removed.

note: isErase will not remove files that are currently in use by other processes.

the error code will be found in errno on failure.

```
int isRename( char *old, char *new );
```

old: original [path]name of an existing isam file
new: new [path]name of isam file

note: isRename will not rename files that are currently in use by other processes.

isRename can be used to move an isam file from one directory to another, but not to a different drive or device.

the error code will be found in errno on failure.

```
int isStart( IsFile *isfd, IsKdsc *kdsc, int len, char *data, int mode );
```

isfd: isam file descriptor
kdsc: description of index to be selected or NULL to retain the current index
len: partial length of key, or 0 for full
data: data record pointer if needed for search mode, or NULL
mode: see search modes

note: this routine has multiple purposes.

if kdsc is passed then the index described will become current.

if length is non zero then isStart will set a partial key length for the selected index, and will only consider the first len bytes of the key during subsequent searches. this will remain in effect until the next isStart call.

the selected location in the index will be flagged as a start point, which means that the next call to isRead, if made with a search mode of ISNEXT or ISPREV, will return the current record, rather than the actual next or previous. this is often useful for initialising next or previous read loops.

after a successful isStart call, isfd->isrecnum will be set to the current record number.

```
int isRead( IsFile *isfd, char *data, int mode );
```

isfd: isam file descriptor
data: pointer to data buffer to be loaded
mode: search mode plus optional lock mode

note: if successful isfd->isrecnum will be set to the current record number.

Ref Files/FAQ's

when isRead is called with ISLOCK, and the record requested is currently locked by another process, isRead will return ISFALSE with an iserrno value of ELOCKED but will still have read the requested record into the data buffer.

```
int isIndex( IsFile *isfd, int index );
```

isfd: isam file descriptor
index: index number to select

note: isIndex is used to select indexes by number. if index is zero then natural order will be selected, otherwise indexes are numbered starting at one for the primary index.

isIndex retains a current record pointer for each index. after selection the current pointer will be at the record selected when the index was last used.

```
int isGoto( IsFile *isfd, long recnum );
```

isfd: isam file descriptor
recnum: an existing active record number

note: this call will locate the specified record in the current index, making it the current index location.

if successful, the specified record image will be found in isam->data.

```
int isData( IsFile *isfd, char *data, long recnum );
```

isfd: isam file descriptor
data: pointer to data buffer to be loaded
recnum: an existing active record number

note: isData performs a direct read on the data file, and does not disturb the current index pointer, and does not check for record locks. it will return ENOREC if the record requested does not exist or has been deleted.

isData is provided primarily as a means of accessing data records in as an efficient a manner as possible.

```
int isWrite( IsFile *isfd, char *data );
```

Ref Files/FAQ's

isfd: isam file descriptor
data: pointer to data record to be written

note: if successful this call will add the data record to the data file and create an appropriate key in all indexes.

int isWrLock(IsFile *isfd, char *data);

note: as for isWrite, but the resultant data record will be locked on successful return.

int isWrCurr(IsFile *isfd, char *data);

note: as for isWrite, but the current pointer in the current index will be set to the record just written.

int isDelete(IsFile *isfd, char *data);

isfd: isam file descriptor
data: data record to be deleted

note: the record image passed is expected to contain enough data to cover the fields required by the primary index.

if the primary index allows duplicates this call will fail, since isDelete has no means of determining exactly which record you are referring to.

the original data image can be found in isfd->data on return.

int isDelCurr(IsFile *isfd);

isfd: isam file descriptor

note: this call will delete the currently selected record.

the original data image can be found in isfd->data on return.

int isDelRec(IsFile *isfd, long recnum);

Ref Files/FAQ's

isfd: isam file descriptor
recnum: record number to be deleted

note: the original data image can be found in isfd->data on return.

int isRewrite(IsFile *isfd, char *data);

isfd: isam file descriptor
data: data image to be updated

note: the record to be updated is determined by the contents of the primary index fields as found in the data image passed.

if the primary index allows duplicates this call will fail.

the original data image can be found in isfd->data on return.

int isRewCurr(IsFile *isfd, char *data);

isfd: isam file descriptor
data: data record to be updated – the contents of the current record in the data file will be replaced with the image passed.

note: the original data image can be found in isfd->data on return

int isRewRec(IsFile *isfd, long recnum, char *data);

isfd: isam file descriptor
recnum: record number to be replaced
data: data image with which recnum is to be replaced.

note: the original data image can be found in isfd->data on return

int isRewNxt(IsFile *isfd, char *data);

isfd: isam file descriptor
data: data record to be updated – the contents of the current record in the data file will be replaced with the image passed.

note: isRewNxt first sets the current pointer to the next record in the current index before updating the original current record.

it is useful in situations where the update would place the

Ref Files/FAQ's

new record between the original current and next keys in the index.

on successful return the application can then read current to obtain the record that was next in the sequence before the update was performed.

an error code of EENDFILE on an unsuccessful return indicates that the update was successful and isRewNxt has reached the end of the file. any other error code indicates a failure in the update operation.

the original data image can be found in isfd->data on return

```
int isLock( IsFile *isfd );
```

isfd: isam file descriptor

note: this call will lock all records in the file against update by any process other than the current, or return ELOCKED if this is not possible.

```
int isUnLock( IsFile *isfd );
```

isfd: isam file descriptor

note: reverses the action of isLock

```
int isRelease( IsFile *isfd );
```

isfd: isam file descriptor

note: this call will release any records that have been manually or automatically locked by the current process.

```
int isRelRec( IsFile *isfd, long recnum );
```

isfd: isam file descriptor

recnum: record number to be unlocked

```
int isRelCurr( IsFile *isfd );
```

Ref Files/FAQ's

isfd: isam file descriptor

note: unlock the currently selected record

```
int isSetUnique( IsFile *isfd, long value );
```

isfd: isam file descriptor

value: new unique value starting point

note: this will set the value returned by the next call to isUniqueID.

```
int isUniqueID( IsFile *isfd, long *value );
```

isfd: isam file descriptor

value: pointer to variable to receive value

note: the value returned is guaranteed unique for the current file.

```
int isIndexInfo( IsFile *isfd, IsKdsc *kdsc, int idx );
```

isfd: isam file descriptor

kdsc: pointer to dictinfo or key description structure

idx: zero for dictinfo, otherwise index number

note: this call serves two somewhat different purposes.

if idx is zero, then it's actually expecting you to pass a pointer to a dictinfo structure, which will then be loaded with information about the current file. refer to isSamInfo below for a somewhat cleaner method.

if idx is non zero then isIndexInfo will return the specified key description, where a value of 1 represents the primary, followed by each successive index in the file.

```
int isSamInfo( IsFile *isfd, struct dictinfo *dict );
```

isfd: isam file descriptor

dict: pointer to dictinfo structure

note: this call will return information about the specified file in the structure passed. the values available are:

Ref Files/FAQ's

```
short di_nkeys; /* number of indexes */  
short di_recsz; /* data record length */  
short di_idxsz; /* index record block size */  
long di_nrecords; /* current record count */
```

```
int isCheckData( IsFile *isfd );
```

isfd: isam file descriptor

note: this routine checks the data file and data file free list for errors, and will return a value with the following bits set in accordance with it's findings:

```
IC_DATREAD 0 /* data file read error */  
IC_BADDATA 1 /* data record corrupt */  
IC_DATAFREE 2 /* data free list is corrupt */
```

if the return value is zero then no errors were detected.

```
int isCheckIndex( IsFile *isfd, int idx );
```

isfd: isam file descriptor

int: key id number, where 1 represents the primary

note: this routine checks the specified index for errors, and will return a value with the following bits set in accordance with it's findings:

```
IC_IDXREAD 3 /* index file read error */  
IC_ORDER 4 /* key out of order */  
IC_COUNT 5 /* index count mismatch */  
IC_BADMAGIC 6 /* bad index node magic number */
```

if the return value is zero then no errors were detected.

```
int isCheckVarlen( IsFile *isam, struct isVarStat *stats );
```

isfd: isam file descriptor

stats: variable length statistics structure, defined in ischeck.h, or NULL to ignore same.

note: this routine checks the variable length storage system for errors, and will return a value with the following bits set in accordance with it's findings:

```
IC_VLMISFILE 12 /* node misfiled in hash table */  
IC_VLHASHLINK 13 /* mangled link in hash list */  
IC_VLMISSING 14 /* nodes missing or unaccounted */
```

Ref Files/FAQ's

```
IC_VLBADDDATA 15 /* data length mismatch */
```

it will also optionally load the following fields in stats:

```
hashcount; /* number of nodes held in hashtable */
hashspace; /* total bytes free space in same */
fullcount; /* number of nodes not in hashtable */
fullspace; /* total (unusable) space in same */
filecount; /* total number of storage nodes in use */
filespace; /* total free space for the file */
```

machine independent data

the library includes the following series of calls to provide a means of generating data that will be transportable to different machines. use of these is only required if you require such portability, since the library will accept native formats without complaint.

```
int ldInt( char *pad );
int stInt( int value, char *pad );
long ldLong( char *pad );
int stLong( long value, char *pad );
int stChar( char *pad, char *str, int len );
int ldChar( char *pad, int len, char *str );
double ldFloat( char *pad );
int stFloat( double value, char *pad );
double ldDbl( char *pad );
int stDbl( double value, char *pad );
int stFltNull( double value, char *pad, short *null );
double ldFltNull( char *pad, short *null );
int stDbINull( double value, char *pad, short *null );
double ldDbINull( char *pad, short *null );
```

in all of the above, st identifies calls that will place a value in the specified location, which can then be retrieved by the matching ld call. the null suffix calls will load the variable passed (by address) in the last parameter with a boolean ISTRUE if the associated value is zero, or ISFALSE if not.

all pads referred to should be the size of the associated data type, with the exception of ldInt/stInt, which refer to a two byte integer, regardless of the native word size.

note that the float and double calls do not perform any actual data conversion, and are provided as a means of avoiding data alignment problems. the int and long calls will perform byte order reversal (sometimes referred to as swabbing) on platforms where the least significant byte is found at the beginning of the field.

note also that ldint and stint operate on short (2 byte) integers, although the value and return are passed in the natural word size.

isam indexes

isam indexes are defined/described by means of the keydesc structure.

the following is an example of the basic procedure, and describes an index that allows duplicates, has full compression, and consists of two fields – a machine independent integer taken from the first two bytes of the record, and a character string from the seventh through sixteenth bytes.

```
struct keydesc key;
```

```
key.k_flags = ISDUPS+ISCOMPRESS; /* see index flags */
key.k_nparts = 2; /* number of fields involved */
key.k_part[0].kp_start = 0; /* offset in data record */
key.k_part[0].kp_leng = INTSIZE; /* length */
key.k_part[0].kp_type = INTTYPE; /* type */
key.k_part[1].kp_start = 7; /* offset in data record */
key.k_part[1].kp_leng = 10 * CHARSIZE; /* length */
key.k_part[1].kp_type = CHARTYPE; /* type */
```

this structure is then passed to isBuild, isAddIndex, or isStart.

index flag values – these apply to the index as a whole

ISNODUPS no duplicates permitted
ISDUPS duplicates permitted
DCOMPRESS compress duplicates
LCOMPRESS leading compression
TCOMPRESS trailing space compression
COMPRESS full compression
TNULL compress on nulls in TCOMPRESS

individual index component/part/field types

CHARTYPE 0 array of bytes/characters
INTTYPE 1 two byte (short) integer
LONGTYPE 2 four byte (long) integer
DOUBLETTYPE 3 ieee double floating point
FLOATTYPE 4 ieee single floating point
MINTTYPE 5 machine (native) short
MLONGTYPE 6 machine (native) long

ISDESC 0x80 add to individual parts for descending order

it is generally advisable to provide some callable means of loading your key descriptions into a keydesc structure, or to provide some other means of easy access, since they will be needed for selecting indexes via isStart.

you will also need to exercise caution when using CHARTYPE fields

Ref Files/FAQ's

as the library does not recognise null terminated strings. in other words, if you're referring to a 10 byte wide character field, and it contains a seven byte string, a null terminator and 2 garbage bytes following the null, the library will build a key 10 bytes in length that includes the null and trailing garbage. this key might prove difficult to locate later..

the officially approved solution is to use the included stchar() and ldchar() calls, which will pad the string passed with spaces on store and replace the null terminator on load.

note also that trailing compression operates on strings of trailing spaces, so the fairly common practice of padding the remainder of the field with nulls could result in inefficient operation where trailing compression is expected.

open modes

there are two different groups of open modes used by the isOpen and isBuild calls. the first group describes the operations which the current process will be allowed to perform, while the latter controls what forms of external access will be permitted. you must chose one from each list.

ISINPUT open file for reading only
ISOUTPUT open file for writing only
ISINOUT open file for both reading and writing
ISSYNCWR all writes flushed to disk before return

ISMANULOCK record locks placed manually
ISAUTOLOCK record locks placed automatically on read
ISEXCLLOCK exclusive access to index and data
ISRONLY read only - no locking

ISVARLEN variable length records - see varlen.ref
ISVRCMP RLE compressed varlen data
ISNOCARE open either fixed or variable length

ISMASKED index masking option - see mask.ref

ISTRANS transaction processing - see trans.ref
ISNOLOG disable logging - see trans.ref

NOTE: ISSYNCWR is only active for systems that permit O_SYNC in the file open call. this precludes most DOS flavoured systems.

search modes

all isRead operations act on the currently selected index.

Ref Files/FAQ's

ISFIRST select the first record in the index
ISLAST select the last record in the index

ISEQUAL search for and return the first exact match, or error
ISGTEQ search for first exact match, or next greater
ISGREAT search for the next greatest key in the index

ISNEXT skip to the next key in the index
ISPREV skip to the previous key in the index
ISCURR reread the current location.

note ISNEXT and ISPREV behave differently when used in isRead immediately after in isStart call in that they will return the current rather than the next/previous record.

this is a feature, not a bug. the intention is to allow isStart to function as an initialisation call preceding a read next/previous loop, as in the following:

```
isStart( isfd, &secondary_index, 0, NULL, ISFIRST );  
while( isRead( isfd, data_pointer, ISNEXT ) )  
display_data( data_pointer );
```

the above will read the contents of isfd from the top in secondary_index order.

lock modes

these are added to the search mode in isRead to lock or wait for the release of the selected record.

ISLOCK lock the selected record against external update
ISWAIT wait until the selected record has been released
ISLCKW wait for and unlock the selected record

ISLOCK will return an ELOCKED error if the selected record has been locked, but will still return the selected record image. this allows the process to read an externally locked record, but warns that updates will not be permitted.

ISWAIT and ISLCKW do no deadlock processing, and will wait forever, so applications should be coded appropriately.

the isLock, isUnLock, isRelease, isRelRec and isRelCurr group of functions provide further control over record locking, and you can also use ISEXCLLOCK to lock other processes out of a file completely.

error codes

Ref Files/FAQ's

in addition to the standard operating system error codes, the library will return the following values in `isfd->iserrno` where appropriate.

EDUPL 100 illegal duplicate
ENOTOPEN 101 file not open
EBADARG 102 illegal argument
EBADKEY 103 illegal key description
EBADFILE 105 isam file is corrupt
ENOTEXCL 106 can't get exclusive access
ELOCKED 107 record is locked
EKEXISTS 108 index already defined
EENDFILE 110 start or end file reached
ENOREC 111 record not found
ENOCURR 112 no current record
EFLOCKED 113 file is locked
EFNAME 114 file name is too long
EBADMEM 116 can't allocate memory
ENOPRIM 127 no primary key

when the library returns a system error code it will also load `isfd->iserrio` with the sum of the following two groups of values:

IO_IDX 0x01 index file
IO_DAT 0x02 data file

IO_OPEN 0x10 while opening
IO_CREA 0x20 while creating
IO_SEEK 0x30 while seeking
IO_READ 0x40 while reading
IO_WRIT 0x50 while writing
IO_LOCK 0x60 while locking

Worldwide Copyright (c) Byte Designs Ltd (2009)

answers to frequently asked questions

answers to frequently asked questions

Q: i'm sure i have compiled the library properly, and i am sure i have the right linker commands, but i keep getting unresolved references to base library (`isUpperCase`) functions..

A: if you're linking wrapper (`islowercase`) calls then it is possible your linker resolves references canonically - in a single pass through the listed libraries. linker objects should be listed in order of precedence - object module(s), wrapper lib, base lib.

Q: everything appears to compile normally, but i sometimes get "FATAL iscomp.c(55)" when trying to read or update my files

A: your compiler is probably defaulting char types to unsigned -

Ref Files/FAQ's

you will need to set your compiler to produce signed char types by default, and rebuild the library.

Worldwide Copyright (c) Byte Designs Ltd (2009)

suggestions, pitfalls and defenses

suggestions, pitfalls and defenses

long duplicate chains

if you are using keys that will result in a large number of duplicate records, you should be warned that deletes and rewrites within these can become painfully slow. this is due to the fact that, when you request an update or delete, the library is forced to start at the beginning of the chain and read sequentially through each looking for the record number you are referring to.

the isam standard does not currently appear to offer a solution to this problem, but if anyone has any suggestions you can be sure we'll be interested.

if this is a concern, specifically during long update runs that will be doing a large amount of duplicate key changes, you might wish to consider deleting the offending indexes before you start, and adding them again after the run is complete. this is potentially dangerous, of course, but the advantages will be significantly faster processing times and better balanced trees on completion.

long duplicate chains are not a problem when writing or reading, since writes get tacked onto the end of the chain, and reads are sequential by definition.

duplicate open files

isam allows you to open the same file many times within a given process. cisam and older disam also allowed some rather dangerous operations on these files. for instance:

```
fd = build file
erase file
write records to fd
close file
```

none of the above will return any errors, but you won't be able to open the file again because it has been erased from the file system – everything written has been discarded.

the ISDUPLOCKS option in isconfig.h, when set to a value of one,

Ref Files/FAQ's

will prevent the library from erasing or renaming files which are already open in the current process.

when ISDUPLOCKS is zero this will not be enforced, and it is up to the application to apply the appropriate degree of caution.

COMPRESS/LCOMPRESS/TCOMPRESS

ISKCOMP 0

due to fixed index file format specifications, a single char is used to store the number of leading compressed bytes and another to store the number of trailing compressed bytes. this forces a limitation of 255 bytes (max unsigned char) for compressed key size, regardless of ISMAXKEY setting to ensure full compression of all trailing spaces or nulls for TCOMPRESS, and all leading duplicate characters for LCOMPRESS.

ISKCOMP 1

allows compressed key sizes larger than 255 bytes, but will only permit partial compression on leading and trailing characters. a maximum of 255 leading duplicate characters, and 255 trailing spaces or nulls will be compressed.

Worldwide Copyright (c) Byte Designs Ltd (2009)

audit trail mechanism

audit trail mechanism

ISAUDIT settings in isconfig.h

- 0 - no audit functionality
- 1 - audit writes, updates and deletes
- 2 - audit writes only
- 3 - audit deletes only
- 4 - audit updates only
- 5 - audit writes and deletes
- 6 - audit writes and updates
- 7 - audit deletes and updates

int isAudit(IsFile *isfd, char *pad, int mode);

isfd isam file descriptor

pad pointer to pad, usage dependent on mode

mode operation mode, see below

note mode is one of the following:

Ref Files/FAQ's

AUDSTART start recording
AUDSTOP stop recording
AUDSETNAME name and path of audit file
AUDGETNAME return current name and path
AUDINFO recording status

start and stop do not refer to pad, setname and getname will store and retrieve (respectively) a null terminated string from or to pad. audinfo will set the first byte of pad to a value of zero (stopped) or one (active).

there are a couple of obvious caveats:

do not start recording until you have set a file name.
be careful not to use the same audit file for more than one file.
the filename cannot be longer than 64 characters.

isAudit generally returns an EBADARG error if you do something wrong.

installation

audit trail support is activated by setting ISAUDIT to a value of 1 in isconfig.h before compiling the library.

notes and caveats

the audit trail is essentially a more streamlined alternative to the full featured transaction processing option. only updates to the file are tracked, and the processing overhead is less. while there is no harm in having both options active in your library at the same time, it would be somewhat redundant to have both in use on the same file.

two things to be aware of:

the audit trail impacts the overall efficiency of the library - if you do not plan to use it, you would probably be best advised to leave it inactive.

once a file has been designated as having an audit trail, and even if the trail is currently in the inactive state (AUDSTOP'ed), updates to the file will be slower because the design (not our fault, talk to the cisam people) requires that the status be checked each time the file is updated by another process. this is only the case when the file is open by multiple processes.

this information presented mostly for reference - the actual impact is reportedly quite minimal under all platforms tested.

the format of the audit trail consists of a header record followed by a data image, repeated sequentially. in the case of variable length files, an additional 2 bytes are included to give the record length. all values are stored in machine independent (ldint/ldlong) format.

```
struct audhead /* audit record header format */
{
char au_type[2]; /* record type (aa/dd/rr/ww) */
char au_time[4]; /* date and time */
char au_procid[2]; /* process id */
char au_userid[2]; /* user id */
char au_recnum[4]; /* record number */
char au_reclen[2]; /* record length if variable */
};

#define AUDHEADSIZE 14 /* audit header length */
#define VAUDHEADSIZE 16 /* if variable length */
```

type codes are translated as follows:

aa – record added to file
dd – record deleted from file
rr – record image before rewriting
ww – record image after rewriting

Worldwide Copyright (c) Byte Designs Ltd (2009)

large files – 64bit implementation

large files – 64bit implementation

as of version 6.12 we have attempted to implement 64bit support for large files. todote most unix platforms as well as NT are supported. as more systems develop 64bit capability we will add the same support for disam. we have attempted to make switching to 64bit support as simple as setting a compile time flag..ISHUGE found in isconfig.h. you may need to redefine the ISHUGE related settings for OFFT and UOFFT, depending on how your OS handles large file support and lseek offsets. second level support can be found in isbase.h where platform specific functions such as lseek64 are defined.

please let us know if you require 64bit functionality for your platform, or if you successfully manage to get 64bit working on any platform not currently setup.

Worldwide Copyright (c) Byte Designs Ltd (2009)

custom key types

custom key types

the following steps are required to add your own custom keys:

set ISCUSTOM to 1 in isconfig.h

set ISCUSTKEYS to 1 in base/iscustom.h, and add defines for the key type(s) and size(s) you require.

add a case for each key type to the switch statement in isCustCmp() in base/iscustom.c. each case should set retc to a negative value if key one is less than key two, positive if greater, and zero if equal.

remove the sample key types unless you need them.

include iscustom in your makefile.

NOTE: be careful, if/when applying updates, that you do not overwrite any changes you have made to the base files.

Worldwide Copyright (c) Byte Designs Ltd (2009)

design concepts

design concepts

in order to facilitate thread safe programming, the core library has been developed to use isam file descriptor structures, rather than the standard integral handle concept. the external function calls are named after the original function names, but use capital letters to keep them separate, as in isOpen rather than isopen. the status and information globals, such as iserrno and iserrio, have been eliminated, and each file descriptor now carries it's own copy.

the standard (lower case) calls are provided as an optional wrapper layer which maintains a dictionary of descriptors to allow mapping via the standard integer file handles. it also defines and passes the standard global variables.

conventions

all public library function names start with 'is' and use capital letters to distinguish keywords.

all structure definitions and typedefs start with a capital letter, and use the same capitalisation rules to distinguish keywords.

the posix standard system calls are wrapped in functions prefixed

Ref Files/FAQ's

by is and an underscore.

local support routines in each module are named in lowercase.

macro definitions are all capitals.

virtually all entry and internal functions return TRUE on success and FALSE on failure. TRUE is defined as 1, FALSE is 0.

buffering

the tree node buffering system operates by handing node images which are no longer part of the current path to a storage mechanism. each index carries it's own path list and storage array.

a single linked list holds the current path and facilitates traversal without having to move data around. the node description structures, which contain current status information and a copy of the node disk image, are are passed to and from the path list and the buffer array as pointers.

the buffer cache is currently optimised on a last in first out basis. it might be worthwhile to consider optimising this further by giving a degree of precedence to nodes from lower levels in the tree, on the assumption that these will take more hits.

buffers are flagged invalid whenever the index file header transaction count has been changed by another process, and will be refreshed as required.

at some point in the future, at the cost of cisam concurrency, it may be worthwhile to consider designing a more detailed currency control which would maintain separate transaction counts for each index, and perhaps additional counts for the index and data free lists, variable length storage space, and so on.

the same buffering system provides a failsafe error recovery method in which updated node images are flagged invalid on error, which then results in a forced refresh before the image can be used again. this method allows for optimistic updates, eliminating the need to check first before embarking on potentially illegal updates. see error handling below for more details.

uncompressed (flat) versus compressed indexes

for the dual purpose of keeping the code cleaner and to improve performance, the index handling routines are split into separate functions where the associated handling is markedly different.

flat indexes, in general, can be operated on mathematically, lend

Ref Files/FAQ's

themselves to shell searches, and permit the current key value to be a simple pointer into the node image.

compressed indexes require that each node be searched in a linear fashion, building the key image in an allocated buffer en route, and generally require a fair amount more processing to operate.

the index handing and manipulation modules – isnode, isgrow and isprune – all adhere to this standard. the decision as to which handling method to apply is made at the top level, eliminating the need for myriad if-then-else tests during processing.

error handling

the new library deals with errors via setjmp/longjmp. each external entry point is responsible for initialising a jump buffer and mode flag in the file descriptor. when an error is detected the thread is passed to an error handling routine which sets the error codes and drops the index locks before returning control the entry call.

the error handling routine is also responsible for clearing the update buffers when triggered in write mode.

the nature of setjmp/longjmp means that it appears to the calling function as if the ISFAIL call, made initially to initialise the jump buffer, had returned TRUE.

Worldwide Copyright (c) Byte Designs Ltd (2009)

disam client/server reference

disam client/server reference

to access isam filesets on remote machines set ISCLIENT in isconfig.h to 1 prior to compiling the Disam library. link your application code on the client to the Disam library as well as to the platforms socket library/libraries (SCO-libsocket.a, SUN-libsocket.a and libnsl.a, NT-wsock32.lib). before executing the application you must start the server program, risamsrv on your server machine. this program may be run in foreground or back ground mode. risamsrv – listens for requests from the client and excutes the isam calls passed from the client program on the isam files on the server machine.

** important: the risamsrv executable expects ISMAXPARTS in isconfig.h to be set to the default setting of 20. If you've compiled your client Disam library with a setting other than 20..please contact Byte Designs so we may customize risamsrv to suit your client library configuration.

there are two ways to let disam know where the remote files exist:

Ref Files/FAQ's

i) specify the ip of the remote machine as well as the full path of the filename when opening the isam fileset.

```
e.g. strcpy( filename, "//209.99.99.9/isampath/testfile" );  
fd = isopen( filename, ISINOUT + ISMANULOCK );
```

-or-

ii) set the environment variable on the client machine to the ip address of the server machine where the isam files reside:

```
RISAMHOST = 209.99.99.9
```

Disam determines if a fileset is local or remote by checking for the prefix of "/" on unix or "\\\\" on dos, win, nt in the filename. if multiple slashes prefix a filename without an ip address..Disam expects RISAMHOST to be set. if no multiple slashes prefix a filename/path Disam defaults to local path/ filename.

as well as the standard group of isam calls, Disam also support remote system calls such as open/sopen, read, write, lseek etc..these are called by prefixing the call with r for example rread()/rwrite().

Disam client/server code defaults the remote port number to 5002. If this port number is already in use you may bypass the default by setting the environment variable RISAMPORT to the port number you wish to use.

Worldwide Copyright (c) Byte Designs Ltd (2009)

dcheck

dcheck

dcheck - isam file integrity check/repair utility

usage - dcheck [-hifbB] [...]

options - h - display isam header information only

i - just check indexes, ignore data file

f - fix corrupt indexes

b - rebuild all indexes

B - rebuild specific index

the option string, preceded by a dash, can be placed anywhere on the command line. all options must be specified in one string, and all options apply to all files specified.

the -B option is a little different. it can be specified multiple times, and each occurrence must be followed by an index number. a value of 1 denotes the primary index, 2 the next, etc. you cannot specify -B on multiple files.

Ref Files/FAQ's

dcheck is not interactive, and will ask no questions, so it can be safely used in batch and script files without operator intervention.

dcheck has been designed to run co-operatively (unless -f or -b specified) on files in active use, but note that other processes will be blocked for the duration of some of the check cycles.

Worldwide Copyright (c) Byte Designs Ltd (2009)

dpack

dpack

dpack - isam file pack/rebuild utility

usage - dpack [...]

dpack takes a list of one or more isam files and will rebuild each in turn. it works by building a temporary copy of the original file and copying all active data records to it, then deleting the original and renaming the working copy.

the temporary file is created in the same directory as the original and is named dppnnnn, where nnnnn is the current process id.

you must ensure that you have write permission on the data directory and that there is enough room for dpack to build the working copy.

dpack will fail if it cannot obtain an exclusive lock on the file.

Worldwide Copyright (c) Byte Designs Ltd (2009)

dreport (new)

dreport

dreport - create .xml, .csv or .txt output from isam data

usage - dreport

dreport takes an .xml format input file that provides the report details, see export/samples directory for sample input files, and produces output based on the input file specifications. dreport allows for isam file cross reference, and filtering of data.

sample tests provided in the export directory.

dschema (new)

dschema

dschema - utility to load, store or dump schema file to/from isam file

usage - dschema [-dDSr]

where d - dump schema layout from isamfile to stdout

D - dump schema layout to path

S - store schemafile to isamfile

r - remove schema layout from isamfile

dschema can be used to store a schema layout for an isam file to the isam file where it can be used to report on data from the isamfile via dreport or simply as a reference to the layout of the isamfile.

dsplit

dsplit

dpack - isam file split utility

usage - dsplit [...]

dpack takes an isam file and will split the original file into two smaller isamfiles, each with the same name as the original for the prefix but with a 1 and 2 appended to the original name. it works by building two temporary copies of the original file and copying half the active data records into filename1 and the second half into filename2. dsplit can be compiled to delete the original files by adding a #define REMOVE to the code before compiling.

the temporary files are created in the same directory as the original.

you must ensure that you have write permission on the data directory and that there is enough room for dpack to build the working copies.

dpack will fail if it cannot obtain an exclusive lock on the file.

dvlreblD (new)

Ref Files/FAQ's

dvlibld

dvlibld - variable length isam file conversion utility specifically for converting variable length isam files from old disam/c-isam standard to new disam v7.0 format - expects ISOLDVARLEN to be set to 0 in isconfig.h prior to compiling library and this utility.

usage - dvlibld

dvlibld converts old disam/c-isam format variable length files to the new variable length format now used by disam. this new format overcomes the old record number limit of 16777216L previously imposed on varinfo and continuation node pointers when being stored in 3 character bytes.

the temporary file is created in the same directory as the original and is named dpxxxxx, where xxxxxx is the current process id.

you must ensure that you have write permission on the data directory and that there is enough room for dvlibld to build the working copy.

dvlibld will fail if it cannot obtain an exclusive lock on the file.

Worldwide Copyright (c) Byte Designs Ltd (2009)

dstats

dstats

dstats - report library statistics - expiration date, compilation date and client limit
usage - dstats

dlist makes use of the isLibInfo() or islibinfo() functions to load and report some library statistics. dstats is only useful if it is recompiled and relinked when ever the library is recompiled, to ensure that the statistics reported are current.

Worldwide Copyright (c) Byte Designs Ltd (2009)

expiration date

expiration date

a simple mechanism for setting an automatic expiration date in the library is provided by means of a define.

isconfig.h ISEXTENDED configuration setting

isconfig.h ISEXTENDED configuration setting

defining ISEXTENDED to 1 will ensure the global ispid will hold the id of the process blocking the lock attempt by the calling process - ISLOCKING 2 only.

Worldwide Copyright (c) Byte Designs Ltd (2009)

isread filtering

isread filtering

this feature is currently only available in the wrapper - please let the support department know if this is a problem to you.

the isfilter() call provides a means of externally controlling which records are accessible via isread(). originally developed inhouse as a means of avoiding the work involved in updating inflexible older application code, it allows the programmer to register a routine that will be called, on a per file basis, to check if each record located by isread() is valid.

```
isfilter( int isfd, int ( *filter )0 );
```

isfd isam file handle
filter application defined filter routine

note once a filter routine has been registered, it will be called automatically each time isread() locates a record().

pass a NULL pointer for filter to deactivate.

the application filter routine should be declared as:

```
int myfilter( int isfd, char *data, int mode );
```

isfd the associated isam file handle
data data record pointer as passed to isread()
mode the mode parameter passed to isread()

if the filter routine returns a value of one, then isread() will return the record found.

if the filter routine sets iserrno and returns a value of zero, then isread() will return the specified error to the application.

if iserrno is not set, and the read mode is one of those listed

Ref Files/FAQ's

below, isread() will attempt to read the next record as shown and, if successful, will call the filter routine again.

ISFIRST read next
ISLAST read previous
ISNEXT read next
ISPREV read previous
ISGREAT read greater
ISGTEQ read greater

isread() will return ENOREC if the original mode was either ISCURR or ISEQUAL.

Worldwide Copyright (c) Byte Designs Ltd (2009)

user info

user info

two functions

```
int isUserInfo( struct IsamFile *isam, int mode, char *pad );  
int isuserinfo( int isfd, int mode, char *pad );
```

where pad is a character array of size USERINFOSIZE.

USERINFOSIZE is fixed at ten bytes and should not be changed.

allow the application to store a fixed length ten byte info pad in the index header of any isam file.

mode has two possible values

ISINPUT load pad from index header
ISOUTPUT write pad to index header

these ten bytes are freely available to any application system that requires the ability to store additional information about a given isam file.

Worldwide Copyright (c) Byte Designs Ltd (2009)

installation

installation

the library is split into two sections – the base and the wrapper – as explained in source.ref. the base source is found in the base subdirectory, and the wrapper source in wrap. all public header files are in the head directory.

Ref Files/FAQ's

if you plan to use only the base library then you can ignore both the wrap directory and head/iswrap.h

if you are planning to use only the wrapped (cisam standard) calls then you should probably compile everything into one library and ignore head/isbase.h

if you intend to use both flavours, then it might be an idea to build two separate libraries, but this is optional.

note that if you plan to use the wrapper calls, or the transaction processing module, the base library **MUST** be compiled with ISADMIN activated.

before starting you will need to edit isconfig.h and adjust it as appropriate – refer to configuration below for details.

NOTE: some compilers produce unsigned char types as the default. this must be changed – at least when compiling the base and wrapper modules – all char types must default to signed. this inconvenience will be corrected in future releases.

your compiler may generate a number of warnings to the effect that certain lines of code have no effect, or that arguments are not used in some functions. this is an expected situation and is due to the fact that turning some options off will result in functions being nulled by means of empty ifdef declarations in some cases, or replaced by empty stubs in others.

the source for the dcheck utility is located in the util directory and should be linked with the base library to produce an executable. documentation for same will be found in dcheck.ref. the same is true for the dpack and dlist utilities.

the test directory contains a collection of benchmark utilities, documented in test.ref

you will find a generic makefile in the make directory. to use the makefile, copy the contents of head, wrap and base into one directory, edit isconfig.h as explained below, edit the makefile and follow the comments.

the makefile should be compatible with most standard unix platforms, and should also be compatible with a fair number of the more standard dos, windows and os2 compilers, as well as the gnu make command.

configuration

```
#define ISADMIN 1 /* global file administration */
```

with this option active, the library will maintain an internal table of active isam file descriptors, reuse system file handles when more

Ref Files/FAQ's

than one instance of the same isam file is opened, and include the isCleanUp call amongst the library entry points. admin is mandatory when the standard wrapper library or transaction processing are used, and definitely advisable if you plan to open more than one instance of the same file at any given time.

```
#define ISAUDIT 1 /* 0:off 1:active */
```

a value of one includes support for cisam standard audit trail logging, see audit.ref for details.

```
#define ISLOGGING 1 /* 0:none 1:standard */
```

a value of one includes support for cisam standard transaction logging and processing, together with a number of enhancements. see trans.ref for more details. NOTE – to use this option you must activate the ISADMIN feature. NOTE also that transaction processing is currently not supported under threads – this will be resolved in the near future.

```
#define ISVARIABLE 0 /* 0:none 1:std 2:pack */
```

a value of one includes support for variable length records in the standard cisam compatible configuration. a value of two provides a better hash table algorithm and finer granularity with regard to the use and reuse of slots in the variable length data storage nodes. both one and two include support for data compression. see varlen.ref for more details.

```
#define ISOLDVARLEN 0 /* 0: new v7.0 format */  
/* 1: old disam/c-isam format */
```

there was a limitation in the old disam/disam96/c-isam variable length in that the maximum record number that could be stored in the 3 char bytes allocated for storing varinfo and continuation node record numbers was 16777216L. this limit has been removed but requires that old files are converted, see dvlreblid.ref via the dvlreblid utility prior to use.

```
#define ISLOCKING 2 /* 0:none 1:old 2:new 3:lck */
```

a value of zero will disable locking entirely, resulting in a single user version of the library. locking one enables the original cisam (prior to version 4) locking algorithm. locking two is the current (standard) method used by cisam versions 4 and greater, may also be faster on some platforms. locking three is compatible with the dos 'third file' locking method, and is provided to allow 'lowest common denominator' NFS concurrency with dos based applications. locking one and three rely on the lockf() system call, and two requires fcntl().

```
#define ISDUPLOCKS 1 /* honour locks in dup opens */
```

cisam allows the same file to be opened more than once in EXCLLOCK

Ref Files/FAQ's

mode within the same process. it also permits the same record to be locked more than one under different filehandles within the same process, not to mention that it is also legal to erase a file that is already open in the current process, after which point any data written to the file will evaporate. set this to zero for cisam compatible operation, but please be careful..

```
#define ISDECLARE 1 /* include prototypes */
```

if non zero then full ansi standard function prototyping is enabled.

```
#define ISDATAVOID 0 /* void data pointers */
```

cisam insists that all record pointers passed must be char type. if ISDATAVOID is set to one, they will be declared as type void which, under some compilers, will allow you to pass anything without warnings. this feature is compiler dependent.

```
#define ISCUSTOM 0 /* custom key types */
```

this option allows the addition of custom key types – refer to custkeys.ref for more details.

```
#define ISBERKELY 0 /* 0:memcpy() 1:bcopy() */
```

a value of one uses the berkely memory handling functions bcopy() and bcmp() and covers for the absence of memset() internally. a value of zero uses the default memcpy(), memcmp() and memset() calls.

```
#define ISLONGID 0 /* allows for long pids */
```

a value of 1 will override the C-ISAM / X/OPEN spec that forces short integer values to be stored in the transaction processing logfile and audit trail file for pids and uids. WARNING: do not set to 1 if you are expecting C-ISAM/X/OPEN concurrency or file compatibility or if the process ids returned from the systems getpid() function are short values.

```
#define ISIDXBLK 1024 /* default index block size */
```

the default index block size to be used when creating isam files. 512 and 1024 are the usual standards. 512 is compatible with older versions of cisam, while 1024 seems to be most common value, and is compatible with current cisam files. 1024 will be enforced when building variable length files, but you can still use other values for your regular data.

```
#define ISDUPLEN 4 /* default duplicate width */
```

Ref Files/FAQ's

the width of the field used to store the duplicate sequence numbers in the index. a value of 4 will permit approximately 2000 million identical keys in any given index, while 2 imposes a limit of just over 32500. 4 is more expensive in terms of disk space, so you may wish to consider using 2 if this is important and you not intend to go over the limit. bear in mind that the limit applies to the max count of an individual identical key, not to the total number of duplicates in the index. 4 is compatible with current cisam, 2 with older versions.

```
#define ISMAXIDX 10 /* indexes per file */  
#define ISMAXPARTS 10 /* parts per key */  
#define ISMAXKEY 128 /* bytes per key */
```

these should be fairly obvious, and refer to the number of indexes per file, the number of parts per index, and the total allowable byte count per index key. the value of ISMAXPARTS is also available via the standard NPARTS define.

```
#define ISMAXBUF 20 /* buffers per index */
```

this is actually a slight misnomer in that the root node is always buffered, so the resultant number of memory allocations will actually be one more than the number you specify here. one important caveat applies – you must specify a value that is at least twice the expected depth of the largest expected index tree. **Note: should you encounter an ISFATAL error: fatal isam error iscomp.c(68) the value of ISMAXBUF should be increased and the entire library recompiled before attempting to continue.

```
#define ISERRBASE 100 /* start error codes at */
```

the library generated iserrno values usually start at 100. on some systems this conflicts with system generated errno values. you can use this define to change the start point.

```
#define ISTHREADED 0 /* threadsafe handling */
```

please refer the thread.ref for details if you plan to use this.

Worldwide Copyright (c) Byte Designs Ltd (2009)

key compression

COMPRESS/LCOMPRESS/TCOMPRESS

ISKCOMP 0

due to fixed index file format specifications, a single char is used to store the number of leading compressed bytes and another to store

Ref Files/FAQ's

the number of trailing compressed bytes. this forces a limitation of 255 bytes (max unsigned char) for compressed key size, regardless of ISMAXKEY setting to ensure full compression of all trailing spaces or nulls for TCOMPRESS, and all leading duplicate characters for LCOMPRESS.

ISKCOMP 1

allows compressed key sizes larger than 255 bytes, but will only permit partial compression on leading and trailing characters. a maximum of 255 leading duplicate characters, and 255 trailing spaces or nulls will be compressed.

Worldwide Copyright (c) Byte Designs Ltd (2009)

transparent file links

transparent file links

when building, opening, renaming or erasing isam files, the library will first check for the existence of a .lnk file in the specified path. this is a text file that is expected to contain the physical path for the associated isam file in the first line. the remainder of the file contents is ignored.

for instance, if you create a file called sample.lnk in directory testdata with "mydata/temp" in the first line, then attempt to open "testdata/sample", the library will bypass any existing sample file in testdata and open mydata/temp instead.

linking is not recursive – the only place the library will look for a link is on the original path passed to the associated isam call.

Worldwide Copyright (c) Byte Designs Ltd (2009)

index masking

index masking

index masking is a mechanism through which the application can control which indexes a given record will be stored in. usage is relatively simple, and best explained by example.

masked files are created by adding ISMASKED to the mode when calling isBuild or isbuild, and are opened in the same way.

the library will return an EBADARG error if a masked file is opened without ISMASKED in the mode or ISMASKED is added to the open mode on an unmasked file.

Ref Files/FAQ's

given a file with three indexes, where a value of 1 represents the primary, 2 the middle index, and 3 the last, the following call will add a record to the primary index only:

```
issetmask( isfd, ISMASK(2) + ISMASK(3) );  
iswrite( isfd, record_data );
```

to rewrite a record such that it will be masked in the primary:

```
issetmask( isfd, ISMASK(1) );  
isrewrite( isfd, record_data );
```

where isrewrite could also be isrewcurr or isrewrec.

the mask value for the current record can be obtained via a call to

```
isgetmask( isfd );
```

record masks must be set explicitly – if reading through a file with the intention of adjusting mask values you must call issetmask for each record you wish to change.

files built with masking active will allow multiple copies of identical indexes to be added to the file. this is to allow the developer to store different record subsets in separate indexes, but it is important to note that only the *last* of these indexes can be accessed via isstart – isindex should be used instead.

BASE: if using the base (file descriptor) calls you should enquire and set the mask values directly, rather than using isgetmask and issetmask. for example:

```
if( isam->mask & ISMASK(1) ) /* current record masked in primary */  
isam->mask = ISMASK(2) + ISMASK(3); /* mask 2nd and 3rd indexes */
```

NOTES: masking is a custom extension – files built with ISMASKED are not isam compatible.

you should not remove indexes from a masked file once it has been populated.

as of version 6.1, the flag in the index header that denotes masked files has been changed – you will need to run fixflags (in the util directory) to patch older files.

see nullkeys.ref for an alternate method of filtering indexes.

Worldwide Copyright (c) Byte Designs Ltd (2009)

isam file mirroring reference

isam file mirroring reference

isam file mirroring works by setting an environment variable BDMIRROR to the full path of location of the mirror fileset.

BDMIRROR will mirror to remote destinations if ISCLIENT is set to 1 in isconfig.h prior to compiling the Disam library. If the mirror files are on a remote machine the path of BDMIRROR must be prefixed with the ip of the remote machine such as:

BDMIRROR = //209.99.99.9/isamfilepath for unix platforms

-or-

BDMIRROR = \\209.99.99.9\isamfilepath for DOS,Win or NT platforms

The mirror fileset must exist in the BDMIRROR location for mirroring to work. All disam calls will be duplicated to the locale fileset as well as the mirrored fileset.

Worldwide Copyright (c) Byte Designs Ltd (2009)

null key masking

null key masking

it is possible to flag indexes such that null keys (those whose value equates to null) will not be added to the index, and the associated records will not be accessible via that index,

the default value used to determine if a given key is null is zero, but this can be modified by adding the value (left shifted by eight) to the key type field.

the following code constructs a null key index in which a character field containing all spaces will be considered null:

```
key->k_flags = ISDUPS + NULLKEY;
key->k_nparts = 1;
key->k_part[0].kp_type = CHARTYPE + ( 32 << 8 );
key->k_part[0].kp_start = 10;
key->k_part[0].kp_leng = 10;
```

null key indexes are not cisam compatible.

see mask.ref for an alternate method of filtering indexes.

Worldwide Copyright (c) Byte Designs Ltd (2009)

pure indexes

pure indexes

pure index files provide a method of managing key information without the overheads required for full data handling. they offer advantages in speed and storage requirements, but are necessarily limited in other ways.

both isBuild and isbuild will create pure index files when passed a value of zero for the record length. the result is a single .idx system file – no .dat is created.

pure index files consist of the primary index only – you may not add secondary indexes.

only the fields pointed to by the key description will be stored and available for retrieval.

the current value of isrecnum will be stored on writes and updates and will be returned on read. since the global isrecnum normally contains the data record number, in the case of pure index files where there are no data records, the following pure idx iswrite will inherit whatever the current isrecnum value is. to allow for the user application setting isrecnum, or to allow the current value to be herited set ISPURE in isconfig.h to 0, this is the current default setting.

if the preference is for disam to assign an incremented isrecnum value prior to iswrite set ISPURE to 1 in isconfig.h

record locking is not possible, and pure indexes may not be opened in autolock mode.

you cannot delete or rewrite by record number.

pure indexes cannot be rebuilt or repaired if damaged, so should not be used for critical data.

Worldwide Copyright (c) Byte Designs Ltd (2009)

auto file corruption repair

auto file corruption repair

index and data file corruptions can occur if a running process is aborted mid-stream or if a system failure occurs.

often running dcheck -b or dpack after such a failure can be time consuming if the effected files are large, not to mention dcheck or dpack must be run manually for each isam file, first to determine the file status and then to repair any corruptions.

by setting ISREPAIR to 1 in isconfig.h prior to compiling the library, Disam will maintain data record images of records being updated, for the duration of the update. if the update concludes successfully the image is removed from the file system, however if a system failure

Ref Files/FAQ's

occurs the data image remains on the file system and Disam uses these images to check and repair the effected index/data records.

this removes the need to rebuild the entire isam file manually, which can be time consuming.

to enable the auto repair facility:

set ISREPAIR to 1 in isconfig.h prior to compiling the Disam library

set the repair file path in isport.h – see #define reppath

this tells Disam where to store the data record image files these files are named as: .rep and are of the format

short int – isam filename/path length

short int – data image length

long int – data record number

char string – isam filename/path

char string – isam data record image

after a system failure all corrupted isam files will be automatically repaired simply by ensuring a call to isStartup() – base, or isstartup() – wrap, is included in the application's main startup program prior to opening any isam files. since only one call to isStartup()/isstartup() is enforced, and since absolutely no isam files should be opened or already open at the time isStartup()/isstartup() is called the application should ensure that only one application routine performs the startup/repair.

the following restrictions apply:

i) at no time should isstartup() be called if other processes are accessing or have open any isam files

ii) only one process should call istartup() immediately after the system has recovered

iii) at no time should the .rep files be manually removed, unless dcheck –b or dpack is used in place of calling isstartup()

Worldwide Copyright (c) Byte Designs Ltd (2009)

DISAMv7.0 schema manipulation routines

DISAMv7.0 schema manipulation routines

int isStSchema(IsFile *isam, char *loadfile)

store schema definitions from loadfile to the isam file

isam isam file descriptor

Ref Files/FAQ's

loadfile file containing a schema layout or a file containing a list of schema layout filenames

if there are multiple record definitions such as is the case with C union usage in record definitions, loadfile can be a file with the names of the different schema layout files

if loadfile is a schema layout DISAM/DB expects the word 'structure' to precede the actual layout, for example:

```
structure
field1 type format length
field2
.
.
```

int isLdSchema(IsFile *isam, char *path)

load schema definitions stored in the isam file to flat files in path

isam isam file descriptor
path path to place schema layout definitions stored in isam file

int isDpSchema(IsFile *isam)

dump any schema layouts stored in isam file to stdout

isam isam file descriptor

int isRmSchema(IsFile *isam)

remove any schema layouts stored in isam file

isam isam file descriptor

Worldwide Copyright (c) Byte Designs Ltd (2009)

status variables

status variables

cisam implements a bank of four char fields named isstat1 through isstat4, primarily for use by isam based COBOL programs.

they are declared and initialised in the wrapper library, but the

code makes no attempt to set or modify the values.

Worldwide Copyright (c) Byte Designs Ltd (2009)

bench test source

bench test source

the test directory contains four stand-alone executables which you may find useful for reliability or performance testing, or as a sample of the standard wrapper interface usage.

all three depend on bench.h and bench.c in the same directory.

auto.c essentially a loose collection of calls that will build, add and delete indexes, write, read and delete pseudo random data records, and check the results. useful for testing the integrity of your compiled library. the standard auto run is 500 records, but this can be changed by specifying a different count on the command line.

lock.c test record locking - will build and populate a sample data file (if not found in current directory) and then loop through the data, locking a sequential subset of the records.

mult.c cyclic add/update/delete test intended for checking the index locking and multi-user features.

user.c an undocumented and simplistic command line interactive test engine used to verify specific operations. it is a long way from complete yet, but might prove useful if you are able to figure it out. the support department may ask you to run it as a means of checking that your compiled library is operating correctly.

user now includes the ability to make system calls (sys) and can run simple scripts (run).

auto and mult will use varlen data if -v is passed on the command line, and will add compression to same if -vc is used.

Worldwide Copyright (c) Byte Designs Ltd (2009)

threadsafe

threadsafe

this feature owes it's existence to Callum Gibson in australia because

without his Expertise and Advice it would never have been completed.

support for multi-threading is provided on two levels. the base level implementation consists of a parallel collection of isam function calls that operate on file descriptor structures (as opposed to the usual integer file handles) and, other than requiring that you do not open the same file more than once in any one process, is fully multi-thread compatible without the requirement for critical section handling.

the second level option provides full cisam standard compatibility and operates on the standard integer based file handles and associated file descriptor table. this option provides critical section (mutex) handling for protection of the descriptor table and logic to provide appropriate handling for duplicate opens on the same file within the same process. the mutex requirement impacts on the efficiency of the library, but allows the use of the standard interface without losing functionality. there are two exceptions.

the four global variables – isrecnum, isreclen, iserrno and iserrio are present and maintained, but should not be considered reliable. instead you should use the following function call equivalents:

```
long *isrecnum( int isfd );  
int *isreclen( int isfd );  
int *iserrno( int isfd );  
int *iserrio( int isfd );
```

note that these functions return pointers rather than values. this allows the application to both enquire and set the value, as in the following examples:

```
if( *is_errno(isfd) == ENOREC ) /* record not found */  
  
*is_recnum(isfd) = saved_record_number;
```

the other exception applies to building variable length files. the standard isbuild call requires the use of a global isreclen value, which is not threadsafe. isvbuild covers this problem by taking the maximum record length as an additional argument:

```
isvbuild( char *name, int fixlen, int maxlen, IsKdsc *key, int mode );
```

NOTE: when using second level support you must call isThreaded() from your root thread before starting any isam threads – this will initialise the master mutex lock.

NOTE: at no time, under any circumstances, should the application allow more than one thread to operate on an individual isam file handle at any given time.

NOTE: isCleanup and iscleanup are inherently not threadsafe – they should only ever be called by the root thread after all isam handling threads have terminated.

NOTE: the transaction processing module is still not threadsafe.

Ref Files/FAQ's

we apologise for the inconvenience and hope to have a solution in the near future.

NOTE: second level threadsafe under os2/warp is only marginally supported. there are a number of limitations imposed by the fact that os2 only provides one critical section lock per thread. at time of writing the library will occasionally deadlock under heavy use. we are researching a solution to this problem but at the same time are hoping that ibm will improve threaded support before it becomes an issue. if you require second level threadsafe operation under os2 please stand up and be counted by contacting the support department.

installation

in order to make use of the base level implementation, the only thing you need to do is to compile a library with the ISADMIN feature turned off. this is because the administration functions make use of global tables which would not be safe in a non-mutexed environment. use of this implementation implies that you will be making base level calls (as detailed in base.ref) and do not intend to allow your applications to open the same file more than once.

to build a second level library you will need to turn the ISADMIN feature on, and make a choice from the four options currently given in isconfig.h as to which ISTHREADED flavour your target will use. so far we have covered SunOS, Solaris, NT and OS2(unstable). we hope to add support for MIT pthreads and the win32 threading extensions soon, and would be most interested to hear from anyone who would be willing to advise on, or help to test these.

if your thread handling library uses a different series of calls than those covered, you might wish to take a look at isthread.h, since the porting procedure is relatively simple. in all cases we will be glad to offer advice in exchange for the opportunity to add more platforms to the list of those supported.

Worldwide Copyright (c) Byte Designs Ltd (2009)

transaction processing

transaction processing

we have attempted to provide a transparent implementation of the transaction processing routines provided by cisam. the library should perform as specified in the cisam manuals, but includes a few additional enhancements.

we are not certain that we have identified the log file locking offsets properly. there is a possibility that the library will not co-operate in concert with cisam, but we have not found any problems to date.

transaction processing is activated via ISLOGGING in isconfig.h

the overlapping transactions option has been modified, and issusplug and isresumlog have been discontinued – see overlapping transactions for more details.

NOTE: please ignore references to multi-thread support in the following – we are currently working on a revision, but this is not yet available. please let support know if this is a problem to you, and we will add you to the list of people to notify when it is ready.

function calls reference

int islogopen(char *name)

opens log file. once the log file is opened all subsequent operations on all files will be logged until the log is closed or suspended. each process is responsible for opening a log file. usually all processes within an application will open the same log file. access is governed by concurrency locks to prevent simultaneous writes, but this feature might not be compatible with cisam. files opened/built with ISNOLOG in the mode will not be logged.

int islogclose(void)

closes log file for the process. all subsequent operations will not be logged.

int isbegin(void)

defines the beginning of a set of operations on all files opened with ISTRANS in the mode argument. this set of operations is known as a transaction.

all records updated within the transaction will remain locked to ensure no other process can affect the results. in multi-user situations it is important to keep the time spent within a transaction to a minimum to avoid impacting on other processes' access to the records involved.

all calls to isclose within the transaction will be deferred to avoid releasing the locks mentioned above. if the same file is reopened within the same transaction you will simply get the same file handle back.

Ref Files/FAQ's

all updates affect the data file immediately. if the process should terminate before completing the transaction, the changes will remain unless isrecover is used to rebuild the data from the last backup.

int iscommit(void)

defines the successful completion of a transaction. all locked records will be released and all deferred closes honoured.

int isrollback(void)

defines the unsuccessful completion of a transaction. all updated records will be restored to their original state, all record locks released and all deferred closes honoured.

int isrecover(void)

used to rebuild a set of data files from an original condition by means of a common log file. islogopen() must be called first to open the log.

usage reference

data file recovery

if, at the time a full data backup was made, the transaction log was purged, then, provided all updating processes call islogopen() before making changes, it is possible to rebuild the data files from the backup and the log file.

it is not necessary that files be opened with a mode of ISTRANS for changes to be logged. files that are opened with a mode of ISNOLOG will not be logged and will not be recoverable.

to do so it is necessary to compile a recovery utility that will call the following functions in order:

```
islogopen();
isrecover();
islogclose();
```

the recovery procedure will then be to restore the backup (make sure not to overwrite the log file) and run the recovery utility.

Ref Files/FAQ's

the utility should check the return from isrecover() to ensure no errors occurred during recovery.

it is also possible to implement incremental backups by this method. the log file can be backed up and purged on a regular basis. to recover it will be necessary to restore the original backup, then restore and run the recovery utility on each log file in chronological order.

transaction processing fundamentals

the theory behind transaction processing is that a collection of changes to a group of data files can be defined in such a way as to be able to undo these changes at any time during the operation.

the collection of changes is referred to as a transaction.

a sample program might best serve to illustrate. proper error checking has been omitted to simplify:

```
/* post a sales invoice */

#include

int invcFile; /* source invoice record file */
int custFile; /* customer master records */
int tranFile; /* accounting transaction file */

main()
{
    islogopen( "logfile" );

    invcFile = isopen( "invoice", INOUT + ISTRANS );
    custFile = isopen( "customer", INOUT + ISTRANS );
    tranFile = isopen( "transaction", INOUT + ISTRANS );

    isstart( invcFile, ..., ISFIRST );

    /* loop through invoices */
    while( isread( invcFile, invcrec, ISNEXT ) == SUCCESS )
    {
        isbegin(); /* define transaction start */

        if(
            postCust() == SUCCESS /* update customer master */
            &&
            postTran() == SUCCESS /* create transaction record */
            &&
            dellInvc() == SUCCESS /* delete invoice record */
        )
            iscommit(); /* successful, commit changes */
        else
            isrollback(); /* failed, erase changes */
    }
}
```

Ref Files/FAQ's

```
}  
  
isclose( invcFile );  
isclose( custFile );  
isclose( tranFile );  
  
islogclose();  
  
exit( 0 );  
}
```

note that the transaction processing logic will lock all records that are changed within the transaction for the duration of the transaction. in some instances it pays to keep the transaction cycle as short as possible to allow other processes access to these records.

note also that those files whose changes are to be erased when isrollback is called must be opened with mode ISTRANS. changes will still be logged for those that are not, but the transaction processing logic ignores them and will not undo any changes.

overlapping transactions

as an extension to the cisam standard, and also as a means of allowing flexible use of transaction processing within multi-threaded applications, the library provides a mechanism for handling multiple concurrent (or overlapping) transactions.

management centers around the use of a callback function provided by the application. in order to permit overlapping transactions you must first call isTxnInit(callback), where callback is the name of a routine that will return a short integer value that the library will use to determine which operations belong to which transaction.

in a single threaded application the simplest method is to declare a variable that is visible to all routines that handle transactions and to the callback function itself.

for example, assuming a simple nested transaction operation:

```
static short txnid;  
  
txnid = 1;  
isbegin();  
for( account = 0; account < 5; ++account )  
{  
  /* update account master files */  
  txnid = 2;  
  isbegin();  
  /* update account files */  
}
```

Ref Files/FAQ's

```
iscommit() or isrollback();  
txnid = 1;  
}  
iscommit() or isrollback();
```

in a multi-threaded application, assuming the intention is to allow atomic transactions within individual threads, you would have the callback routine return the thread id value.

NOTE: isrecover is NOT threadsafe and should only be used under single threaded circumstances, or at very least only when there are no other threads working on isam files.

NOTE: when implementing transaction processing in a client/server environment the logfile MUST be opened as:
isLogOpen/islogopen("//ip/path/logname");

Worldwide Copyright (c) Byte Designs Ltd (2009)

variable length records

variable length records

refer to install.ref for activation and configuration details.

the following is a list of the functional and programmatic differences between fixed length and variable length isam file handling.

isreclen is a global integer used to pass variable length info around in much the same way as isrecnum.

isaddindex

the parts of any key added must reside within the fixed length part of the record.

isbuild

you can add ISVARLEN to the mode argument to create a varlen file. if so then the record length argument must define the maximum possible length of the records. you must also set isreclen to the size of the fixed length header portion of the record. key fields can only reside in the fixed length portion. you can also use ISVARCMP in place of ISVARLEN to specify that RLE compression be applied to the data.

isvbuild

isvbuild should be used in place of isbuild if you are creating variable length files in a multithreaded environment - refer to thread.ref for details.

isindexinfo

if file is variable, the msb of di_nkeys will be set to indicate same. di_recsz contains the maximum record size, the fixed length size is returned in isreclen.

Ref Files/FAQ's

isopen

if the file is variable length you must use ISVARLEN in the mode, or else isopen will return an EBADARG error. the converse is also true - if the file is not variable length, then specifying ISVARLEN will return the same error. if successful then isreclen will be set to the maximum record length of the opened file.

to get around the inconvenience of the above, you can use ISNOCARE in the open mode, in which case the file will be opened regardless of whether it is variable or fixed length. you can then check the open mode value to determine which kind of file was opened. this is found in isam->openmode if you are using the base library calls, or loaded by means of isgetmode(int isfd, int *mode) if using the wrapper library.

isread

isreclen will be set to the actual size of the record just read.

isrewcurr

isrewrec

isrewrite

iswrcurr

iswrite

set isreclen to actual size of record before calling.

compression

the library provides optional RLE compression on the variable length portion of the data record. this is specified when creating the file by using a mode flag of ISVARCMP in place of ISVARLEN. note that ISVARCMP and ISVARLEN are NOT additive - you must use one or the other, not both.

once a compressed varlen file has been created, you can use either ISVARLEN or ISVARCMP when opening the file - either will work, and the file will be opened in the appropriate mode.

note that RLE encoding is only valuable if your data has a fair distribution of repeated zero (0), decimal zero (48), or space (32) bytes.

compression is transparent to the application - records are written and read as before.

compression is not cisam compatible.

notes

the check utility will analyse and test the variable length storage nodes and report statistics.

Worldwide Copyright (c) Byte Designs Ltd (2009)

wrapper reference

wrapper reference

the standard interface is provided in the form of a wrapper, and is intended to allow plug and play compatibility with existing cisam (or equivalent) application code. only the differences between the base and wrapper interfaces are explained below – you should refer to the base library reference for the remaining details.

the integer handle wrapper

this interface is provided for those who prefer or require a cisam compatible API. this interface will link transparently with existing cisam or cisam compatible application source, and should require no changes to your code, other than to switch to the iswrap.h header.

all modules that use the integer wrapper should include iswrap.h

all function names are identical except that they use exclusively lower case letters.

all parameters are identical except that an integer file handle is used in place of the IsamFile structure pointer, and except for the isbuild() call, which has the standard cisam arguments, whereas the base version includes an extra argument for variable length support.

all functions return 0 on success, or -1 on error, with the exception of isbuild and isopen, which return an integer file handle or -1 on error.

SUCCESS and ERROR are defined in the header and can be used to test the returns from these calls.

isrecnum, isreclen, iserrno and iserrio are global variables declared in the wrapper, defined as externals in iswrap.h and are maintained in the same way as defined for the equivalent isfd values in the base, and by the usual cisam standard.

since the base library maintains an individual set of these variables on a per file basis, we have also provided a collection of four calls that will return a pointer to the associated variable:

```
long *is_recnum( int isfd );  
int *is_reclen( int isfd );  
int *is_errno( int isfd );  
int *is_errio( int isfd );
```

Ref Files/FAQ's

these can be used to enquire or set values for individual files, and are not overwritten on each isam call made.

common dictionary access

char *isdi_name(int isfd); file name as used in isopen/isbuild
int isdi_datlen(int isfd); fixed record length
int isdi_curidx(int isfd); current index number (zero based)
int isdi_idxfd(int isfd); index file system handle
int isdi_datfd(int isfd); data file system handle
struct keydesc *isdi_kdsc(int isfd); current key description

isdi_name return a pointer to the file name string, or NULL if the isfd value is invalid. the remainder return an integer value or -1 on failure.

Worldwide Copyright (c) Byte Designs Ltd (2009)
