

dbi Group

John D. Baker

October 10, 2020

Contents

dbi Overview	2
dbi Interface	2
Running dbi	2
dbi Source Code	6
=: Index	45

dbi Overview

dbi is a J script that reads and write inverted .dbi files.

dbi is described in the blog post [APL Software Archaeology .dbi Edition](#).

- <https://analyzethedatanotthedrive1.org/2013/12/26/apl-software-archaeology-dbi-edition/>

dbi source is available here:

- <https://github.com/bakerjd99/jacks/blob/master/dbi/dbi.ijs>

dbi Interface

```
dbicreate    [11] create dbi file
dbiread      [21] read dbi file
dbitemplate  [25] (x) argument for (dbicreate) from dbi file
dbiwrite     [27] write field data to dbi file
dbimetadata  [15] extracts dbi file metadata
```

Running dbi

Using the J `dbi.ijs` script is simple. The included file `allnsf.dbi` is an example .dbi file that holds all the *field types* supported by `dbi.ijs`. Simple fields are similar to SQL database columns and result in single column tables when fetched. Repeated fields are actually tables and result in tables when fetched. All fields are stored as contiguous byte runs in the .dbi file making access as fast as it's ever going to get.

The `.dbi` format works very well for numeric (integer and floating point) vectors and tables. The `.dbi` byte representation matches native host formats and can be easily read and written by other programs. Simple ASCII character data is also supported but the `.dbi` format is not text oriented.

To run download the [GitHub dbi directory](#) to a local directory. I placed the files in `c:/temp`.

Load `dbi.ijs`:

```
load 'c:/temp/dbi.ijs'
```

View the file template:

```
dbitemplate 'c:/temp/allnsf.dbi'
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|ALLNSF|u1f|u21u1|u4f|u17u4|u8f|u37u8|i16f|i16i16|i32f|i20i32|f64f|f97f64|d6f|c0|c15|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|1713  |U1 |21U1 |U4 |17U4 |U8 |37U8 |I16 |16I16 |I32 |20I32 |F64 |97F64 |D6 |C0|C15|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

This file has 1713 *records*. Read all the fields:

```
d=. dbiread 'c:/temp/allnsf.dbi'
```

The result is a two row boxed table. The first row contains field names and the second row holds the field data.

```
80 list 0 { d NB. dbi field names
ALLNSF_u1f    ALLNSF_u21u1  ALLNSF_u4f    ALLNSF_u17u4  ALLNSF_u8f
```

```
ALLNSF_u37u8  ALLNSF_i16f  ALLNSF_i16i16 ALLNSF_i32f  ALLNSF_i20i32
ALLNSF_f64f  ALLNSF_f97f64 ALLNSF_d6f  ALLNSF_c0  ALLNSF_c15
```

```
80 list datatype > 1 { d NB. fields have various J datatypes
boolean boolean integer integer integer integer integer integer
integer integer floating floating integer literal literal
```

Indirect assignment is a handy way to extract all the fields as nouns:

```
(0{d)=: 1{d NB. assign values in 1 to names in 0
```

```
80 list 'A' nl ''
ALLNSF_c0  ALLNSF_c15  ALLNSF_d6f  ALLNSF_f64f  ALLNSF_f97f64
ALLNSF_i16f  ALLNSF_i16i16 ALLNSF_i20i32 ALLNSF_i32f  ALLNSF_u17u4
ALLNSF_u1f  ALLNSF_u21u1 ALLNSF_u37u8 ALLNSF_u4f  ALLNSF_u8f
```

Creating and writing a new .dbi is straight forward.

```
t=. dbitemplate 'c:/temp/allnsf.dbi'
```

Start the new file with 0 records. The system preallocates file space if the record count is not zero.

```
tc =. (<":0) (<1;0)} t
```

NB. nonzero result indicates success

```
tc dbicreate 'c:/temp/allcopy.dbi'
```

2048

NB. write all fields - result is byte count

```
d dbiwrite 'c:/temp/allcopy.dbi'
```

1782243

Read the copy back and compare data:

```
dc =.dbiread 'c:/temp/allcopy.dbi'
```

```
(1{d) -: 1{dc
```

1

The `.dbi` system can read and write single fields and supports datetime data. See code comments for more details.

John Baker; *original: December 26, 2013; revised: October 9, 2020*

dbi Source Code

```
NB.*dbi s-- create/read/write APL inverted dbi files.
NB.
NB. interface word(s):
NB.
NB. dbicreate      NB. create dbi file
NB. dbimetadata   NB. extracts dbi file metadata
NB. dbiread       NB. read dbi file
NB. dbitemplate   NB. (x) argument for (dbicreate) from dbi file
NB. dbiwrite      NB. write field data to dbi file
NB.
NB. created: 2012mar27
NB. -----
NB. 12oct09 (jodliterate) group documentation added
NB. 14jul04 (bytebits) renamed (bytebits3) with result note

coclass 'dbi'

NB.*end-header

NB. base dbi field types: no repetitions or scale factors
DBIBASETYPES=: <;._1 ' U1 U4 U8 I16 I32 F64 D6 C0'

NB. special APL name characters in 0 QuadAV of source APL
DBISPECIAL=: 145 95 241
```

NB. version level of dbi files - last APL version is 3.00

DBIVERSION=: 3

NB. interface words (IFACEWORDSdbi) group

IFACEWORDSdbi=: <.;_1 ' dbicreate dbiread dbitemplate dbiwrite dbimetadata'

NB. root words (ROOTWORDSdbi) group

ROOTWORDSdbi=: <.;_1 ' IFACEWORDSdbi ROOTWORDSdbi dbicreate dbiread dbitemplate dbiwrite'

NB. applies the verb in string (x) to (y)

apply=: 128!:2

NB. signal with optional message

assert=: 0 0"_ \$ 13!:8^:((0: e.])^` (12"_))

NB. boxes open nouns

boxopen=: <^:(L. = 0:)

NB. like ((8\$2) #: a. i.]) but always returns boolean

bytebits3=: _8 {."1 [: 2&#.0(0 1&i.)^:_1 a. i.]

changestr=: 4 : 0

*NB.*changestr v-- replaces substrings - see long documentation.*

NB.

NB. dyad: clReps changestr cl

```

NB.
NB.  NB. first character delimits replacements
NB.  '/change/becomes/me/ehh' changestr 'blah blah ...'

pairs=. 2 {"(1) _2 [\ <; _1 x      NB. change table
cnt=_1 [ lim=. # pairs
while. lim > cnt=.:cnt do.        NB. process each change pair
  't c'=. cnt { pairs            NB. /target/change
  if. +./b=. t E. y do.          NB. next if no target
    r=. I. b                     NB. target starts
    'l q'=. #&> cnt { pairs      NB. lengths
    p=. r + 0,+/\(<:# r)$ d=. q - 1 NB. change starts
    s=. * d                      NB. reduce < and > to =
    if. s = _1 do.
      b=. 1 #~ # b
      b=. ((1 * # r)$ 1 0 #~ q,l-q) (,r +/ i. l)} b
      y=. b # y
      if. q = 0 do. continue. end. NB. next for deletions
    elseif. s = 1 do.
      y=. y #~ >: d r} b        NB. first target char replicated
    end.
    y=. (c $~ q *# r) (,p +/i. q)} y NB. insert replacements
  end.
end. y                            NB. altered string
)

d6=: 3 : 0

```



```
NB.*d6 v-- convert 7 column integer array timestamps to 6 bytes.  
NB.  
NB. monad: cl =. d6 itT7
```

```
'not 7 item timestamps' assert isd6 y  
,ts6Frts7 y  
)
```

```
dbicheckdata=: 4 : 0
```

```
NB.*dbicheckdata v-- tests field name/data table.  
NB.  
NB. At least one field must exist, field record counts (first  
NB. axis) must all match and all field datatypes must match  
NB. expected datatypes from file.  
NB.  
NB. If the data passes all tests the result of this verb is a  
NB. permutation list that sorts the data table into file field  
NB. order. Any test failure throws an assertion as these errors  
NB. cannot be safely ignored.  
NB.  
NB. dyad: ilOrd =. btData dbicheckdata btMetadata
```

```
msg=. 'invalid name data table'  
msg assert (2=#$x) *. (2={.$x) *. (1<:{:$x) *. isboxed x  
msg assert (*. /ischar&> 0{x) *. 1 = # ~. #&> 1{x
```

```
NB. (-.)=: !(*)=. dbiname fir nrf nff fnm fnc fty fnb fsc fts fbd
```

```
(0{y})=. 1{y
```

NB. field names must match names in file

```
pfx=. <dbiname, '_'  
dfnm=. pfx ,&.> fnm  
msg assert (0{x) e. dfnm
```

NB. fields to file order

```
ord=. /:dfnm i. 0{x  
x=. ord {"1 x
```

NB. mask of fields in data

```
fmsk=. dfnm e. 0{x
```

NB. null fields are considered type correct

```
if. 0=nrf do. ord  
else.
```

NB. check column counts of any repeated fields

```
ccnt=. {:@$&> 1{x  
fnc=. fmsk#fnc  
rmsk=. 0 < fnc
```

```
msg assert (rmsk#fnc) = rmsk#ccnt  
ccnt=. rmsk * #@":&> <"0 ccnt
```

```
ety=. fmsk #"1 }."1 dbifieldtypes y  
ety=. (pfx ,&.> 0{ety) (,0)} ety
```

```

ety=. (((<'is') ,&.> tolower&.> ccnt }.&.> 1{ety) (,1)} ety

NB. character Cn fields require custom type
NB. tests other types have (isType) verbs
if. +./mc=. ('isc'&-:)@ (3&{.)&> 1{ety do.
  NB. NIMP only basic char tests for now
  (mc # 1{ety)=. iscfld
end.

NB. apply type tests - they're very fussy and
NB. are driven by internal noun representations
NB. ie: if your integers are currently floats
NB. these type tests will fail. It's the caller's
NB. job to get basic J types right
'invalid field type(s)' assert (1{ety) apply&> 1{x
ord
end.
)

dbicreate=: 4 : 0

NB.*dbicreate v-- create dbi file.
NB.
NB. dyad:  iaBytesAlloc =. btNamesTypes dbicreate clPathFile
NB.
NB.  ntab=. <;._1 ' 93 I32 F64 30I16 30U1 D6 C0 C15'
NB.  ntab=. (;:'TEST COUNT HEIGHT SCORE BITS DATE NOTE MESSAGE') ,: ntab
NB.  ntab dbicreate 'c:\temp\test.dbi'

```

```
NB.  
NB.  NB. clone existing file  
NB.  tgdbi=. 'c:\temp\test.dbi' [ srdbi=. 'c:\temp\classes.dbi'  
NB.  (dbitemplate srdbi) dbicreate tgdbi  
NB.  (dbiread srdbi) dbiwrite tgdbi  
  
'dbi file exists' assert -.fexist y  
'dbi extension missing' assert 'dbi' -: tolower justext y  
msg =. 'invalid name type table'  
msg assert (2=#$x) *. (2={.$x) *. (1<{.$x) *. isboxed x  
msg assert (*. / 1 >: ,(#@$)&> x) *. *. /, ischar&> x  
dbinamecheck 0{x  
  
NB. (-.)=: !(*)=.  fnb fnc fty fsc  
(0{dtty)=. 1{dtty=. dbiparsetypes }. 1{x  
  
NB. records and fields  
msg assert 0<:nrf=. _1&". ;0{1{x  
nff=. <:{$x  
  
NB. total size and field offsets  
fbd=. dtty dbioffsets nrf,nff  
fbd=. }.fbd [ nbf=. 0{fbd  
  
NB. set remaining header items - generate file header  
dbiname=. ;(<0;0){x  
fir=. DBIVERSION
```

```
fts=. nff # ,:t7stmp 6!:0''
fnm=. }. 0{x
dm=.      dbiname;fir;nrf;nff;fnm;fnc;fty;fnb;fsc;fts;fbd
dm=.  (,:'dbiname fir nrf nff fnm fnc fty fnb fsc fts fbd'),:dm
fhd=. dbiheader dm

NB. create file - write header
('unable to write header -> ',y) assert 0<:bytes=. fhd fwrite y

NB. extend file for data
bytes=. (nbf,bytes) fresize y
)

dbifieldtypes=: 3 : 0

NB.*dbifieldtypes v-- field types from metadata.
NB.
NB. monad:  bt =. dbifieldtypes btMetadata

NB. (-.)=: !(*)=. dbiname fir nrf nff fnm fnc fty fnb fsc fts fbd
(0{y)=. 1{y

NB. repetition counts
msk=. fnc > 0
trc=. msk #^:_1 ":%> msk # <"0 fnc

NB. field repetitions and types
tft=. <"1 trc ,. fty ,. ":% ,. fnb
```

NB. scale factors

```
msk=. fsc > 0
tft=. tft ,&.> msk #^:_1 msk # '.'&,@":&.> <"0 fsc
```

NB. dbname and record cnt

```
|:((dbname ; ":nrf) , fnm ,. tft) -.&.> ' '
)
```

```
dbiheader=: 3 : 0
```

*NB.*dbiheader v-- format dbi metadata header.*

NB.

NB. monad: dbiheader btMetadata

NB.

NB. dbiheader dbimetadata 'C:\BCA\bcadev\CA\ULTCL.DBI'

NB. (-.)=: !()=. dbname fir nrf nff fnm fnc fty fnb fsc fts fbd*

```
(0{y)=. 1{y
ni32=. nff,4
```

```
t1=. (nff,128)$0{a.
```

NB. initial descriptors

```
t1=. (16 {."1 >fnm) (<a;;i. 16)} t1
```

NB. field names

```
t1=. (ni32$2 ic fnc) (<a;;16 + i.4)} t1
```

NB. field repetitions I32

```
t1=. (ni32$2 ic fnb) (<a;;20 + i.4)} t1
```

NB. field lengths I32

```
t1=. fty (<a;;24)} t1
```

NB. field types

```
t1=. (fsc{a.) (<a;;25)} t1
```

NB. field scale factors

```
t1=. (ts6Frts7 fts) (<a;;26 + i.6)} t1
```

NB. field time stamps

```

t1=. (ni32$2 ic fbd) (<a;;32 + i.4)} t1      NB. byte offsets to fields I32
t1=. ' ' , t1                                NB. file header
t1=. (4{.'DBI') (<0;i. 4)} t1                NB. .DBI signature
t1=. ('4.2' (;@ (8!:0)) fir) (<0;4 + i. 4)} t1 NB. version tag
t1=. (8{.dbiname) (<0;8 + i. 8)} t1          NB. table name
t1=. (2 ic nrf,nff) (<0;16 + i.8)} t1       NB. number of records & fields

```

```

,t1 NB. cl result
)

```

```

dbimetaddata=: 3 : 0

```

```

NB.*dbimetaddata v-- extracts dbi file metadata.

```

```

NB.

```

```

NB. monad: bt =. dbimetaddata clPathFile

```

```

NB.

```

```

NB. dbimetaddata 'C:\BCA\bcadev\TX\ULTCL.DBI'

```

```

NB. dbimetaddata 'C:\BCA\bcadev\CA\classes.dbi'

```

```

NB. read 128 byte header

```

```

('to small for a .dbi file -> ',y) assert 128<dsiz=. fsize y

```

```

emsg=. 'unable to read file -> ',y=. utf8 y

```

```

emsg assert _1 -.@-: hdr=. iread y;0 128

```

```

NB. record count, field count

```

```

'nrf nff'=. _2 ic (16 + i. 8){hdr

```

```

hdrsize=. 128 * >:nff

```

```

'file size header mismatch' assert hdrsize<:dsiz

```

NB. field descriptors

```
emsg assert _1 -.@-: fdsc=. iread y;128,nff*128
dbiparseheader hdr,fdsc
)
```

```
dbinamecheck=: 3 : 0
```

*NB.*dbinamecheck v-- check dbi table and field names.*

NB.

NB. The tests applied here are stricter than the original APL

NB. functions. APL characters, including the underbar character,

NB. are not allowed and lengths are enforced.

NB.

NB. monad: dbinamecheck blclNames

```
alpha=. ((65+i.26),97+i.26){a.
```

```
digits=. '0123456789'
```

NB. first name is table name

```
dbn=. ;0{y
```

```
'invalid table name' assert -.(({.dbn) e. digits) +. (8 < #dbn) +. 0 e. dbn e. alpha,digits
```

```
fnm=. }. y
```

```
'field names not unique' assert ~:fnm
```

```
msg=. 'invalid field name(s)'
```

```
msg assert -. 1 e. ({.&> fnm) e. digits
```

```
msg assert fnm *./@:e.&> <alpha,digits
```



```
msg assert 16 >: #&> fnm
```

```
1 NB. names ok  
)
```

```
dbioffsets=: 4 : 0
```

```
NB.*dbioffsets v-- compute byte offsets to fields and total dbi bytes.
```

```
NB.
```

```
NB. dyad: il =. bt dbioffsets ilNrfNff
```

```
NB. (-.)=: !(*)=. fnb fnc fty fts fbd
```

```
(0{x)=. 1{x
```

```
'nrf nff'= . y
```

```
NB. bytes for field data - no scale factors are _1 in (fnc)
```

```
fbnew=. >.(nrf * ((fty e. 'CD'){1 8) * (|fnc) * 1 >. fnb) % 8
```

```
NB. preserve lengths of any extant c0 fields
```

```
if. 5 = {:$x do.
```

```
  if. #pos=. I. 0 = fnb do. fbnew=. (pos { }. (fbd,0) - 0,fbd) pos} fbnew end.  
end.
```

```
NB. total bytes and offsets in file to field data
```

```
_1 |. (128 * nff + 1) + +/\ 0,fbnew
```

```
)
```

```
dbiparseheader=: 3 : 0
```

```
NB.*dbiparseheader v-- parses dbi file header.
NB.
NB. monad: bt =. dbiparseheader clDbiHeader

fdsc=. 128}.y [ hdr=. 128{.y NB. (-.)=:

'not a .dbi file' assert 'DBI ' -: 4{.hdr
'.dbi version <: 2.00' assert 2.00 < fir=. _1&".(4 + i. 4){hdr

dbiname=. ;dbirepsnc <toupper ((8+i. 8){hdr) -. ' '
if. *./'STFREQ' E. dbiname do. dbiname=. 'STFREQ' end.

'nrf nff'=. _2 ic (16 + i. 8){hdr
fdsc=. (nff,128)$fdsc
fnm=. dbirepsnc (<"1 (i. 16) {"1 fdsc) -.&.> ' ' NB. field names

NB. field repeat cnts (I, U and F only) <0 field is list; >:0 field is fnc column table
fnc=. _2 ic ,(16 + i.4) {"1 fdsc

NB. field types (C character, I signed integer, U unsigned integer, F floating, D date+time)
fty=. 24 {"1 fdsc

NB. field lengths (bits for I, U, F; bytes for C, D)
fnb=. _2 ic ,(20 + i.4){ "1 fdsc

NB. scale factors (base 10; for U, I fields only)
```

```
fsc=. a. i. 25 {"1 fdsc
```

```
NB. decode 6 byte/48 bit field time stamps
```

```
fts=. ts7Frts6 (26 + i.6) {"1 fdsc
```

```
NB. byte offsets in file to field starts
```

```
fbf=. _2 ic ,(32 + i.4) {"1 fdsc
```

```
NB. result table - names match APL functions
```

```
desc=. ;:'dbiname fir nrf nff fnm fnc fty fnb fsc fts fbd'  
desc  ,: dbiname;fir;nrf;nff;fnm;fnc;fty;fnb;fsc;fts;fbd  
)
```

```
dbiparsetypes=: 3 : 0
```

```
NB.*dbiparsetypes v-- checks and parses field types.
```

```
NB.
```

```
NB. monad: bt =. dbiparsetypes blclFtypes
```

```
NB.
```

```
NB. NB. all types in all fields in all dbi's in directory
```

```
NB. ftypes=. {"1 ; }.&.> dbitemplate&.> 1 dir 'C:\BCA\bcadev\CA\*.DBI'
```

```
typ=. 'CIUFD' [ msg=. 'invalid field types'
```

```
m0 =. y e.&.> <typ
```

```
NB. field types
```

```
msg assert 1 = +/&> m0
```

```
NB. must occur exactly once
```

```
fnc=. - {.&> m0
```

```
NB. defaults non-repeated fields _1
```

```

NB. extract any repetitions
if. #t0=. ,(_1&".)@,&> ,m0 <;._2&> y do.
  m1 =. fnc ~: _1
  msg assert 0<:t0=. m1#t0          NB. must be numeric >: 0
  fnc=. t0 (I. m1)} fnc            NB. all repetitions
end.

fty=. ,m0 #&.> y                  NB. remove any repetitions
y=. ,m0 <;._1&> y
t0=. y i.&.> '.'

NB. numeric types without repetitions and scale
NB. factors are limited to DBIBASETYPES
t1=. fty ,&.> t0 {.&.> y
fty=. ;fty
msg assert (('C'~: fty)#t1) e. DBIBASETYPES
msg assert _1<fnc=. _1&".&> t1 -.&.> <typ

NB. extract any scale factors
t1=. t0 }.&.> y
if. 1 e. m0=. 0<fsc=. #&> t1 do.
  msg assert 'UI' e.~ m0 # fty    NB. only U I fields scale
  t0=. I. m0
  msg assert 1='.' +/@:=&> t0{y    NB. only one '.' is valid
  t1=. _1&".&> (t0{t1) -.&.> '.'
  msg assert -. _1 e. t1          NB. scale is numeric
  fsc=. t1 t0} fsc

```

```
end.
```

```
(;:'fnb fnc fty fsc') ,: fnb;fnc;fty;fsc NB. (-.)=:  
)
```

```
dbiread=: 3 : 0
```

```
NB.*dbiread v-- read dbi file.
```

```
NB.
```

```
NB. monad: bt =. dbiread clPathFile
```

```
NB.
```

```
NB. NB. all fields in file order
```

```
NB. dbiread 'C:\BCA\bcadev\TX\ULTCL.DBI'
```

```
NB.
```

```
NB. dyad: bt =. blclFieldnames dbiread clPathFile
```

```
NB. bt =. clFieldName dbiread clPathFile
```

```
NB.
```

```
NB. NB. unboxed argument is one field
```

```
NB. 'INDGRP' dbiread 'C:\BCA\bcadev\CA\classes.dbi'
```

```
NB.
```

```
NB. NB. read many fields
```

```
NB. (;:'PPINDEX LOSS') dbiread 'C:\BCA\bcadev\TX\ULTCL.DBI'
```

```
0 dbiread y
```

```
:
```

```
'dbi file does not exist' assert fexist y
```

```
NB. (-.)=: !(*)=. dbiname fir nrf nff fnm fnc fty fnb fsc fts fbd
```

```
(O{dm)=. 1{dm=. dbimetadata y

nbf=. fsize y=. utf8 y    NB. bytes in file
nrf=. nbf dbitestnrf dm   NB. test record count
fbdx=. fbd,nbf            NB. extended offsets
fty=. tolower fty

if. x-:0 do.
  rdr=. lff=. i. nff      NB. read all fields
else.
  rf=. boxopen x          NB. read at least one field
  'invalid field names' assert rf e. fnm
  lff=. fnm i. rf
  rdr=. rf i. lff{fnm     NB. requested field order
end.

NB. read data by inverted field
emsg=. 'unable to read file -> ',y
dat=. i.0
for_iff. lff do.
  n1=. iff{fnb            NB. field length
  t=. n2=. iff{fnc         NB. field repetition count
  n2v=. (n2 >: 0) # n2     NB. repetition count as list
  t=. (t < 0){t,1         NB. list field repetition count is 1
  s1=. iff{fsc            NB. field scale factor

  if. nrf=0 do. t=. ''    NB. no records
```

```

else.
  NB. base and byte length of field
  l1=. ((>:iff){fbdx) - b1=. iff{fbd
  emsg assert _1 -. @-: t=. iread y;b1,l1
end.

NB. convert bytes to field types
select. iff{fty
case. 'f' do.
  'floating point field must be 64 bits' assert n1 e. 64
  t=. _2 fc t
  if. n2 >: 0 do. t=. (nrf,n2)$t end.
case. 'i' do.
  'signed integer must be 16 or 32 bits' assert n1 e. 16 32
  t=. (-n1%16) ic t
  if. n2 >: 0 do. t=. (nrf,n2)$t end.
  if. s1 ~: 0 do. t=. t % 10~s1 end.
case. 'u' do.
  'unsigned integer field must be 1, 4 or 8 bits' assert n1 e. 1 4 8
  if. n1=1 do. t=. (nrf*1>.n2) {. ,bytebits3 t
  elseif. n1=4 do. t=. dfb ((*nrf,n2v),4)$,bytebits3 t
  elseif. n1=8 do. t=. a. i. t
  end.
  if. n2 >: 0 do. t=. (nrf,n2)$t end.
  if. s1 ~: 0 do. t=. t % 10~s1 end.
case. 'd' do.
  'date fields must be 6 bytes' assert n1 e. 6

```

```
t=. ts7Frts6 (nrf,6)$t
case. 'c' do.
  NB. variable length leading char delimited or fixed length
  if. n1=0 do. t=. ];._1 t else. t=. (nrf,n1)$t end.
end.
dat=. dat,<t
end.

NB. name value table in requested order - (dbiname) has no blanks
(,rdr) {"1 (2,#rdr) $ ((<dbiname,'_') ,&.> lff{fnm) , dat
)
```

```
dbirepsnc=: 3 : 0
```

```
NB.*dbirepsnc v-- replace special APL name characters.
NB.
NB. Replace special APL characters that can occur in dbi field
NB. and table names, eg: delta, underbar, underbar_delta with hex
NB. encoded versions with layout h $\alpha$ HH. The J dbi interface will
NB. not create dbi files with these characters in names but will
NB. tolerate their presence in dbi files created elsewhere. The
NB. hex value HH depends on the QuadAV encoding of the creating
NB. APL. For APL+WIN these characters have postions (QuadIO = 0)
NB. of (DBISPECIAL=: 145 95 241) or in hex 91 5F F1
NB.
NB. monad: bl =. dbirepsnc blclName
```



```
(<spcrep DBISPECIAL) changestr&.> y
)
```

```
dbitemplate=: 3 : 0
```

```
NB.*dbitemplate v-- (x) argument for (dbicreate) from dbi file.
```

```
NB.
```

```
NB. Create template for file (y) which if used as a left (x)
```

```
NB. argument to (dbicreate) would clone the file structure and
```

```
NB. size.
```

```
NB.
```

```
NB. monad: bt =. dbitemplate clPathFile
```

```
NB.
```

```
NB. dbitemplate 'C:\BCA\bcadev\CA\classes.dbi'
```

```
'dbi file does not exist' assert fexist y
```

```
dbifieldtypes dbimetadata y
```

```
)
```

```
dbitestnrf=: 4 : 0
```

```
NB.*dbitestnrf v-- test record count against field lengths and return correct value.
```

```
NB.
```

```
NB. dyad: iaNrf =. iaBytes dbitestnrf btMetadata
```

```
NB. (-.)=: !(*)=. dbiname nrf nff fnm fnc fty fnb fsc fts fbd
```

```
(0{y)=. 1{y [ nbf=. x
```

NB. follows APL function

```
n2=. 1 (I. 0>fnc)} fnc
```

```
v1=. fnb = 0
```

NB. number of bits for this field (0 for C0 fields)

```
n2=. fnb * n2 * (fty e. 'CD'){1 8
```

NB. number of bytes for this field (0 for C0 fields)

```
n1=. >.(nrf * n2)%8
```

NB. actual field lengths

```
t1=. ((}.fbd),nbf) - fbd
```

```
if. #v1x=. I. v1 do. n1=. (v1x{t1) v1x} n1 end.
```

```
if. n1 -. @-: t1 do.
```

NB. field lengths do not match expected lengths

NB. fields with computable record counts (all but C0,U1,U4 unless 8nU1 or 2nU4)

```
if. +./v1=. (fty e. 'IFD') +. (fty e. 'CU') *. n2 >: 8 do.
```

NB. one or more fields have computable record counts

NB. record count in data

```
t2=. (8 * v1#t1) % v1#n2
```

NB. field record counts must match

NB. NIMP: there's a bug here - the source

NB. APL function wouldn't work either

NB. 'field length error' assert 1 = #~.t2

```
    NB. computed record count - take most frequent length
    NB. this is the best guess until the calc bug is resolved
    mode2 <.t2
  end.
else. nrf
end.
)
```

```
dbiwrite=: 4 : 0
```

```
NB.*dbiwrite v-- write field data to dbi file.
NB.
NB. dyad: (ctNames ,: blulData) dbiwrite clPathFile
NB.
NB.  dfile=. 'C:\temp\ULTCL.DBI'
NB.  fdat=. dbiread dfile
NB.
NB.  NB. double data - full file rewrite
NB.  fdat=. (0{fdat) ,: (1{fdat) ,&.> 1{fdat
NB.  fdat dbiwrite dfile
NB.
NB.  NB. grab any two fields - replace field data
NB.  fdat=. (2 {. ?~ {:$fdat) {"1 fdat
NB.  fdat dbiwrite dfile
```

```
'dbi file does not exist' assert fexist y
```

```
NB. (-.)=: !(*)=. dbiname fir nrf nff fnm fnc fty fnb fsc fts fbd
(0{dm)=. 1{dm=. dbimetadata y=. utf8 y
ord=. x dbicheckdata dm
x=. ord {"1 x NB. fields to file order

if. nrf = {.#&> 1{x do.
  NB. record count matches file replace selected fields
  (0;y;<dm) dbiwritefields x
else.
  NB. record count differs from file rewrite entire file
  NB. dbi files make no attempt to manage file allocations
  NB. the dbi format is a packed data format.
  'all fields must be specified if record count does not match file' assert ({$x) = nff
  (1;y;<dm) dbiwritefields x
end.
)

dbiwritefields=: 4 : 0

NB.*dbiwritefields v-- write dbi fields.
NB.
NB. dyad: iaBytes =. (paReset;clPathFile;<btMetadata) dbiwritefields btData

'rwf dfile dm'=. x

NB. (-.)=: !(*)=. dbiname fir nrf nff fnm fnc fty fnb fsc fts fbd
(0{dm)=. 1{dm
```

NB. update metadata

```
nrf=. #>0{1{y
fbd=. (,:'fnc fty fsc fbd') ,: fnb;fnc;fty;fsc;fbd
fbd=. fbd dbioffsets nrf,nff
fbd=. }.fbd [ nbf=. 0{fbd
fts=. nff # ,:t7stmp 6!:0''
fir=. DBIVERSION
dm=.      dbiname;fir;nrf;nff;fnc;fty;fnb;fsc;fts;fbd
dm=. (,:'dbiname fir nrf nff fnc fty fnb fsc fts fbd'),:dm
```

NB. reset file for complete rewrites

```
bc=. 0 [ emsg=. 'unable to write dbi file -> ',dfile
if. rwf do. emsg assert 0<:bc=. (dbiheader dm) fwrite dfile end.
```

NB. write field data

```
if. 0=nrf do. bc
else.
```

```
lff=. fnm i. (>:#dbiname) }.&.> 0{y
'full rewrite needed for C0 fields' assert -(0 e. lff{fnb) *. nff~:{$y
fty=. tolower fty
```

NB. expected field data lengths

```
elen=. (}.fbd,nbf) - }.fbd,nbf
hup=. -.rwf [ hlen=. 128 * >: nff
```

```
for_iff. lff do.
```

```

dat=. >iff_index{1{y
select. iff{fty
  case. 'f' do. dat=. f64 dat
  case. 'i' do.
    dat=. ,dat
    if. 0<iff{fsc do. dat=. dat * 10^iff{fsc end.
    dat=. i16`i32 @.(<:(iff{fnb)%16) dat
  case. 'u' do.
    dat=. ,dat
    if. 0<iff{fsc do. dat=. dat * 10^iff{fsc end.
    dat=. u1`u4`u8 @.(1 4 8 i. iff{fnb) dat
  case. 'd' do. dat=. d6 dat
  case. 'c' do.
    if. 0=iff{fnb do.
      NB. (255{a.) does not occur in
      NB. field data at this stage
      dat=. ;((255){a.) ,&.> rtrim&.> <"1 dat
      NB. variable length char field lengths are not
      NB. known until now - requires a header update
      elen=. (#dat) iff} elen
      fbd=. }:hlen , hlen + +/\ elen
      hup=. 1
    else.
      dat=. ,(iff{fnb) {."1 dat
    end.
end.
NB. update field timestamps for partial writes

```

```

    fts=. (t7stmp 6!:0 '') iff} fts
    NB. insure actual and expected lengths match
    emsg assert (iff{elen) = #dat
    emsg assert _1 -. @-: dat iwrite dfile; iff{fbd
    bc=. bc + dat=. #dat
end.
NB. variable length C0 character fields alter offsets
if. hup do.
    dm=. (fts;fbd) (<1;(0{dm) i. ;:'fts fbd'}) dm
    emsg assert _1 -. @-: (dbiheader dm) iwrite dfile;0
end.
bc
end.
)

```

```

NB. decimal from binary: dfb 1 0 0 1 1
dfb=: 2&#. @ (0 1&i.)

```

```

f64=: 3 : 0

```

```

NB.*f64 v-- convert floating array to double 64 bit binary.
NB.
NB. monad: cl =. f64 fu

```

```

'not f64 floating' assert isf64 y=. ,y
2 fc y
)

```

NB. boxes UTF8 names

`fboxname=: ([: < 8 u: >) ::]`

NB. float/character conversion

`fc=: 3!:5`

NB. 1 if file exists 0 otherwise

`fexist=: 1:@(1!:4) ::0:@(fboxname&>)@boxopen`

`fresize=: 4 : 0`

*NB.*fresize v-- resize file.*

NB.

NB. This verb simulates the {Quad}nresize function in APL+WIN.

NB.

NB. NIMP: the winapi directly supports file resizing. Replacing

NB. this function with direct (cd) winapi calls is an option if

NB. performance becomes an issue.

NB.

NB. verbatim:

NB.

NB. BOOL WINAPI SetEndOfFile(__in HANDLE hFile);

NB.

NB. in Kernal32.dll

NB.

NB. dyad: iaSize fresize clPathFile


```
'nbf bytes'=.x

if. nbf=bytes do. nbf
else.
  msg=. 'unable to resize file'

  NB. slow but safe method
  NB. msg assert 0<((0{a.)#~ nbf - bytes) fappend y

  NB. fast and not so safe
  NB. J indexed writes (on windows) can occur beyond
  NB. eof this has the effect of extending the file
  NB. no data is written and when read the unset regions
  NB. are all null (0{a.) characters.
  msg assert _1 -.@-: ' ' iwrite y;<:nbf

  NB. check actual size against expected/required size
  msg assert 0<fbytes=. fsize y
  msg assert fbytes = nbf
  fbytes
end.
)

NB. size of file in bytes
fsize=: 1!:4 ::(_1:)@(fboxname&>)@boxopen

NB. write to file (UTF8 file names) returns #bytes if successful _1 otherwise
fwrite=: ([ , []) (#@[ [ 1!:2) ::(_1:) [: fboxname ]
```

NB. hex from decimal: hfd 5078

```
hfd=: 16&#. @('0123456789ABCDEF'&i.)^:_1
```

```
i16=: 3 : 0
```

*NB.*i16 v-- convert integer array to sets of 2 bytes.*

NB.

NB. monad: cl =. i32 iu

```
'not i16 integer' assert isi16 y=. ,y
```

```
1 ic y
```

```
)
```

```
i32=: 3 : 0
```

*NB.*i32 v-- convert integer array to sets of 4 bytes.*

NB.

NB. monad: cl =. i32 iu

```
'not i32 integer' assert isi32 y=. ,y
```

```
2 ic y
```

```
)
```

NB. integer/character conversion

```
ic=: 3!:4
```

NB. indexed file read - returns cl bytes if successful _1 otherwise

```
iread=: 1!:11 ::(_1:)
```

NB. tests for boxed data

isboxed=: 32&=@(3!:0)

iscfield=: 3 : 0

*NB.*iscfield v-- basic type test for character field.*

NB.

NB. monad: pa =. iscfield uu

NB. (255{a.}) is an expected APL delimiter. The APL

NB. functions do not check for this character in

NB. character data - if present it will break the

NB. the field record count.

if. ischar y do. -. (255{a.}) e. y else. 0 end.
)

NB. tests for character data

ischar=: 2&=@(3!:0)

isd6=: 3 : 0

*NB.*isd6 v-- 1 if (y) is representable as 6 byte timestamps 0*

NB. otherwise.

NB.

NB. Each row of (y) is a 7 integer timestamp: yy,mn,dy

NB. hr,nm,sc,mss where the last item (mss) is decimal

```
NB. milliseconds.
NB.
NB. monad: pa =. isd6 uu

if. -. (isint y) *. (2=#$y) *. (7={:$y) do. 0
elseif. (1 7$0) -: ~.y do. 1 NB. special case all "zero" dates
elseif.do.
  NB. check timestamps
  if. 0 e. valdate 3 {"1 y do. 0
  elseif. 0 e. (0&<: *. 24&>:) 3 {"1 y do. 0
  elseif. 0 e. (0&<: *. 60&>:) 4 5 {"1 y do. 0
  elseif. 0 e. (0&<: *. 999&>:) 6 {"1 y do. 0
  elseif.do. 1
  end.
end.
)

NB. 1 if (y) is f64 representable 0 otherwise
isf64=: 8&=@(3!:0)

isi16=: 3 : 0

NB.*isi16 v-- 1 if (y) is (signed) i16 representable 0 otherwise.
NB.
NB. monad: pa=. isi16 uu

if. isint y=. ,y do. */1=(_32769 32767) I. (<./ , >./) y else. 0 end.
)
```

NB. 1 if (y) is (signed) i32 representable 0 otherwise

`isi32=: isint`

NB. tests for nonextended integers - booleans are considered integers

`isint=: 1 4 e.~ 3!:0`

NB. 1 if (u) is u1 representable 0 otherwise

`isu1=: 1&=@(3!:0)`

`isu4=: 3 : 0`

*NB.*isu4 v-- 1 if (y) is u4 representable 0 otherwise.*

NB.

NB. monad: pa =. isu4 uu

`if. isint y=. ,y do. */((<./ , >./) y) e. i. 16 else. 0 end.
)`

`isu8=: 3 : 0`

*NB.*isu8 v-- 1 if (y) is u8 representable 0 otherwise.*

NB.

NB. monad: pa =. isu8 uu

`if. isint y=. ,y do. */((<./ , >./) y) e. i. 256 else. 0 end.
)`

NB. indexed file write

```
iwrite=: 1!:12 ::(_1:)
```

NB. extracts the extension from qualified file names

```
justext=: '"_`([ #~ [: -. [: +./\.' '&=)@.(' '&e.)
```

```
mode2=: 3 : 0
```

*NB.*mode2 v-- finds the most frequently occurring item(s) in a
NB. list.*

NB.

NB. monad: ul =. mode2 ul

NB.

NB. mode2 ?.500#100

NB. mode2 ;:'I do what I do because I am what I am'

```
if. 0 < # y =. ,y do.      NB. null lists have no modes
  f =. #/.~ y              NB. nub frequency
  (~. y) #~ f e. >./ f     NB. highest frequency items
else. y
end.
)
```

NB. trim right (trailing) blanks

```
rtrim=: ] #~ [: -. [: *./\.' '"_ = ]
```

NB. form special character replacements

```
spcrep=: [: , ('/' ,. [: ,. a. {~ ]) ,. '/hx' ,"1 [: hfd ,
```

```
NB. format 7 integer item timestamp yr mn dy hr mn ss mss
t7stmp=: [: <. ] , 1000 * 1 | {:
```

```
tolower=: 3 : 0
```

```
NB.*tolower v-- convert to lower case.
```

```
NB.
```

```
NB. monad: cl =. tolower cl
```

```
x=. I. 26 > n=. ((65+i.26){a.) i. t=. ,y
($y) $ ((x{n) { (97+i.26){a.) x}t
)
```

```
toupper=: 3 : 0
```

```
NB.*toupper v-- convert to upper case
```

```
NB.
```

```
NB. monad: cl =. toupper cl
```

```
x=. I. 26 > n=. ((97+i.26){a.) i. t=. ,y
($y) $ ((x{n) { (65+i.26){a.) x}t
)
```

```
ts6Frts7=: 3 : 0
```

```
NB.*ts6Frts7 v-- 6 byte representation from 7 integer column timestamp.
```

```
NB.
```

NB. This verb is the inverse of (ts7Frts6) it packs 7 integer timestamps into 6 bytes.

NB.

NB. monad: ctByte6 =. ts6Frts7 itTimestamp7

NB. clByte6 =. ts6Frts7 ilTimestamp7

NB.

*NB. t7=. (<.6 {. ts) , <.1000 * 1 | {:. ts=. 6!:0''*

NB. t7 -: ts7Frts6 ts6Frts7 t7

NB. (10#,:t7) -: ts7Frts6 ts6Frts7 10#,:t7

NB. NIMP better check 7 column timestamps

'invalid 7 integer timestamps' assert 7 = {:\$y

r=. }:\$y

*z=. ((*r),48)\$0*

z=. ((12#2) #: 0 {"1 y) (<a.;i. 12)} z NB. years

z=. ((4#2) #: 1 {"1 y) (<a.;12 + i. 4)} z NB. months

z=. ((5#2) #: 2 {"1 y) (<a.;16 + i. 5)} z NB. days

z=. ((5#2) #: 3 {"1 y) (<a.;21 + i. 5)} z NB. hours

z=. ((6#2) #: 4 {"1 y) (<a.;26 + i. 6)} z NB. minutes

z=. ((6#2) #: 5 {"1 y) (<a.;32 + i. 6)} z NB. seconds

z=. ((10#2) #: 6 {"1 y) (<a.;38 + i. 10)} z NB. milliseconds

NB. convert to character and shape

*(r,6)\$(dfb ((8 %~ #z),8)\$z){a. [z=. ,z
)*

ts7Frts6=: 3 : 0

*NB.*ts7Frts6 v-- 7 integer column timestamp from 6 byte representation.*

NB.

NB. monad: it =. ts7frts6 ctByte6

NB. bytes to 0 1 boolean array - each byte expands to 8 bits

tsb=. , "2 bytebits3 y

NB. year(12), month(4), day(5), hour(5), minute(6), milliseconds(10)

|: dfb@|:> (1 (+/\0 12 4 5 5 6 6)} 48#0) <|.1 |: tsb

)

u1=: 3 : 0

*NB.*u1 v-- convert boolean array to u1 character.*

NB.

NB. Packs 8 bits into bytes. If not divisble by 8 last

NB. 8 bits (last byte) are zero filled in right most bits.

NB.

NB. monad: cl =. u1 pu

'not u1 boolean' assert isul y=. ,y

*a. {~ dfb _8]\ (8 * >.(#y)%8) {. y*

)

u4=: 3 : 0

*NB.*u4 v-- convert integer array to 4 bit unsigned integers.*

NB.

NB. Packs two integers in (i.16) into one byte.

NB.

NB. monad: cl =. u4 iuInt

'not u4 integer' assert isu4 y=. ,y

n=. >. 0.5 * # y

y=. (n,2) \$ (2 * n) {. y *NB. pair items (zero if odd)*

(16 16 #. y){a. *NB. pairs to 0-255 char indexes*

)

u8=: 3 : 0

*NB.*u8 v-- convert integer array to 8 bit unsigned integers.*

NB.

NB. monad: cl =. u8 iuInt

'not u8 integer' assert isu8 y=. ,y

y{a.

)

NB. character list to UTF-8

utf8=: 8&u:

valdate=: 3 : 0

*NB.*valdate v-- validates lists or tables of YYYY MM DD Gregorian
calendar dates.*

NB.

NB. monad: valdate il/it

NB.

NB. valdate 1953 7 2

NB. valdate 1953 2 29 ,: 1953 2 28 NB. not a leap year

```
s=. }:$y
```

```
'w m d'=. t=. |:(*/s),3)$,y
```

```
b=. */(t=<.t),(_1 0 0<t),12>:m
```

```
day=. (13|m){0 31 28 31 30 31 30 31 31 30 31 30 31
```

```
day=. day+(m=2)*-/0=4 100 400|/w
```

```
s$b*d<:day
```

```
)
```

NB.POST_dbi post processor.

```
smoutput IFACE=: (0 : 0)
```

NB. (dbi) interface word(s):

NB. =====

NB. dbicreate NB. create dbi file

NB. dbimetadata NB. extracts dbi file metadata

NB. dbiread NB. read dbi file

NB. dbitemplate NB. (x) argument for (dbicreate) from dbi file

NB. dbiwrite NB. write field data to dbi file

```
)
```

```
cocurrent 'base'  
coinsert  'dbi'
```

Index

apply, 7
assert, 7

boxopen, 7
bytebits3, 7

changestr, 7

d6, 8
DBIBASETYPES, 6
dbicheckdata, 9
dbicreate, 11
dbifieldtypes, 13
dbiheader, 14
dbimetadata, 15
dbinamecheck, 16
dbioffsets, 17
dbiparseheader, 17
dbiparsetypes, 19
dbiread, 21

dbirepsnc, 24
DBISPECIAL, 6
dbitemplate, 25
dbitestnrf, 25
DBIVERSION, 7
dbiwrite, 27
dbiwritefields, 28
dfb, 31

f64, 31
fboxname, 32
fc, 32
fexist, 32
fresize, 32
fsize, 33
fwrite, 33

hfd, 34

i16, 34

i32, 34
ic, 34
IFACE, 43
IFACEWORDSdbi, 7
iread, 34
isboxed, 35
iscfield, 35
ischar, 35
isd6, 35
isf64, 36
isi16, 36
isi32, 37
isint, 37
isu1, 37
isu4, 37
isu8, 37
iwrite, 38

justext, 38

mode2, 38

ROOTWORDSdbi, 7
rtrim, 38

spcrep, 38

t7stmp, 39
tolower, 39
toupper, 39
ts6Frts7, 39
ts7Frts6, 40

u1, 41
u4, 41
u8, 42
utf8, 42

valdate, 42