Wang Laboratories, Inc.

To: CS/2200/386 file
From: TBO

Date: March 14, 1990..

Re: Description of CS/2200/386 disk catalog area stored program file structure.

Disclaimer of Warranties and Limitation of Liabilities

WANG LABORATORIES, INC.
One Industrial Avenue, Lowell, MA 01851 TEL. (508) 459-5000, Telex 172108.

Re: Description of CS/2200/386 disk catalog area stored program file structure.

Index

```
Catalog index area:  contained beginning in disk sector 0.

Disk sector 0  Bytes 0-15.
               .0.1 .2.3 .4.5 .6.7 .8.9 .A.B .C.D .E.F
     example:  0146 448F FE00 0000 0000 0000 0000 0000
    on D31 index sectors = 70'  end cat.area =65023  Current end =17550

    where byte  00 is the index type
                    00 = old index structure eg. default
                    01 = new index structure eg. ' hash.
          byte  01 is number of index sectors (binary count 01-FF).
          bytes 02-03 are 2 byte binary pointer to Current end + 1.
          bytes 04-05 are 2 byte binary pointer to End of the catalog area + 1.
          bytes 06-15 are set 00    (currently undefined).

The remainder of the catalog area is divided into 16 byte file items.
               .0.1 .2.3 .4.5 .6.7 .8.9 .A.B .C.D .E.F
     example:  1080 00FA 00FF 0000 544C 5853 5441 5254
               Active Program   "TLXSTART" on disk at 250 - 255

    where byte  00 is file status.
                    00 = unassigned.
                    10 = active
                    11 = scratched
                    21 = re-used scratch entry.
          byte  01 is file type.
                    00 = data file.
                    80 = program file, "old" format. i.e. MVP
                    40 = program file, "new" format, i.e. 386.
          bytes 02-03 are 2 byte binary pointer to file START address.
          bytes 04-05 are 2 byte binary pointer to file END address.
          bytes 06-07 are set 0000  (currently undefined).
          bytes 08-15 are the file name in ascii.
```

Hashing algorithms:

"The new index structure uses a more efficient hashing algorithm for locating files in the index, resulting in a more even distribution of file entries. Additionally, if an index sector is full, the system enters a new file into the next higher sector rather than the next lower, as is done with the old index. "
Access to the new Disk Index (from SRN for MVP Release 2.5.)
The following program simulates the new disk index hashing:

```
100 N$=F$:REM F$ contains the name of the file to be entered.
110 X$=HEX(00)
120 FOR I=1 TO 8
130  X1$=STR(N$,I,1):REM X1$ contains Ith byte of the file name
140  IF MOD(I,2)=0 THEN 160
150  ROTATE (X1$,4) : REM. Exchange upper/lower nibbles of odd bytes
160  X$=X$ ADD X1$  : REM X$ contains sum of all bytes
170 NEXT I
180 S = MOD(VAL(X$),C):REM C contains the number of index sectors.
190 REM S contains the sector number into which the entry should go.
```

The following program allows access to either the old or the new index algorithm.

```
0010 DIM Z9$(16)16,A$80,B$1                          Nodified 3/13/90.

6500 DEFFN'229(Z9,STR(A$,1,8))                  # ref,   FileName
    : DATA LOAD BA T #Z9,(0,Z3)Z9$()            Read cat sector zero
    : Z4=VAL(STR(Z9$(),2,1))                    Z4=
    : STR(A$,9,8)=STR(A$,1,8)                   Set file name into w/a
    : ON POS(HEX(00 01)=STR(Z9$(),,1))GOSUB 6770,6800   go do hashing logic.
    : Z5=Z3                                     Set should be hash sector in index.
    : IF Z3=0 THEN 6630                         avoid duplicate read of sector 0.
6620 DATA LOAD BA T #Z9,(Z3,Z8)Z9$()            Read index sector.
6630  Z6=0
    : FOR Z7=1TO 16
    : IF Z3 <> 0 THEN 6680
    : IF Z7=1 THEN Z7=2
6680 B$=STR(Z9$(Z7),,1)
    : ON POS(HEX(00 10 11)=B$)GOTO 6740,6720,6720
    : GOTO 6750

6720 IF STR(Z9$(Z7),9,8) <> STR(A$,1,8) THEN 6750
    : Z6=Z7
6740 Z7=16
6750 NEXT Z7
    : IF B$=HEX(00) THEN RETURN
    : IF Z6 <> 0 THEN RETURN
    : B$=HEX(00)
    : IF V9 <>0 THEN 6760
    : Z3=Z3-1
    : IF Z3=Z5 THEN RETURN
    : IF Z3<0 THEN Z3=Z4-1
    : GOTO 6620

6760 Z3=MOD(Z3+1,Z4)
    : IF Z3<>Z5THEN 6620
    : RETURN

6770 XOR(STR(A$,10,7),STR(A$,9,8))                   Old method hash logic.
    : B$=STR(A$,16,1)
    : STR(A$,18,2)=HEX(00 00)
    : ADDC(STR(A$,18,2),B$)
    : ADDC(STR(A$,18,2),B$)
    : ADDC(STR(A$,18,2),B$)
    : ADD(STR(A$,18,1),STR(A$,19,1))
    : Z3=VAL(STR(A$,18,2))
    : Z3=Z3-INT(Z3/Z4)*Z4
    : RETURN

6800 STR(A$,18,1)=HEX(00)                            New method hash logic.
    : FOR Z5=9 TO 16
    : IF MOD(Z5,2)>0 THEN ROTATE(STR(A$,Z5,1),4)
    : STR(A$,18,1)=STR(A$,18,1) ADD STR(A$,Z5,1)
    : NEXT Z5
    : Z3=MOD(VAL(STR(A$,18,1)),Z4)
    : RETURN
```

Storage of CS/2200/386 program files on disk media is as follows:

1. *an item in the disk catalog index area, described on prior pages.*
2. *A program file stored on contiguous sectors within the disk catalog area.*
    - a. *Header block - one sector at the beginning of the file.*
    - b. *Program record blocks.*
    - c. *Trailer block - one sector at the end of the file.*
    - d. *End of File block - written at the maximum numbered sector.*

Header block.

```
Byte  001 - 001 ( 1 byte )  = 40  Program file, VLSI normal or wrap mode .
Byte  001 - 001 ( 1 byte )  = 50  Program file, VLSI scrambled mode.
Byte  001 - 001 ( 1 byte )  = 60  Program file,  386 normal or wrap mode .
Byte  001 - 001 ( 1 byte )  = 70  Program file,  386 scrambled mode .

Bytes 002 - 009 ( 8 bytes ) FileName  1-8 ASCII characters with trailing spaces
Byte  010 - 010 ( 1 byte )  = FD
Bytes 011-256  undefined  usually 00.
```

Program record blocks.

The content of program record blocks differs according to the program SAVE or RESAVE method. Details of each storage method method are detailed on the following pages.

Program trailer blocks.
The content of program trailer blocks differs according to the program SAVE or RESAVE method. Details of each storage method method are detailed on the following pages.

End of file block.
The End of File block is written in the maximum numbered sector of the assigned file space. The format is:

```
    20xx yy followed by all hex 00's.
```

A program file may, on option, be saved with a Date /Time stamp of the form:.

```
    20xx yy00 0000 0001 2033 2D31 342D 3930 ........ 3-14-90
    2031 323A 3031 0000 0000 0000 0000 0000  12:01.........
          followed by all hex 00's.
```

Selecting the option Date/Time stamping of program files.

With CS/386 release 1.0B and greater program files have the option of being written with a Date/Time stamp. A statement of the form :

```
    SELECT T OFF    deactivates writing/listing the Date/Time stamp.
    SELECT T ON     activates writing/listing  the  Date/Time stamp.
```

The default for the Date/Time stamp is the SELECT T OFF condition.

Program file structure : CS/2200 MVP normal mode.

Method invoked by default mode, or "SELECT OLD" followed by SAVE or RESAVE. The "text line" content is a combination of textatoms and ascii text, refer to the table "BASIC-2 Verb atoms".

.MVP normal mode -- program sectors.

```
40 FileName FD 00 ... 00                                Header record
00 FF hhhh text line 0D 0000                            Program record
   FF hhhh text line 0D 0000
   ...
BASIC-2 text atom tables.

   FF hhhh text line 0D 0000 FD

00 FF hhhh text line 0D 0000                            Program record
   FF hhhh text line 0D 0000
   ...
   FF hhhh text line 0D 0000 FD

20 FF hhhh text line 0D 0000                            Trailer record
   FF hhhh text line 0D 0000
   ...
   FF hhhh text line 0D 0000 FE
```

.MVP wrap mode -- program sectors.

```
40 FileName FD 00 ... 00                                Header record
00 FF aa FFA0 bbbb
       cc hhhh text line statement
       cc hhhh text line statement
00 FF continued text line statement
       cc hhhh text line statement
20 FF continued text line statement
       cc hhhh text line statement
```

```
where     aa  is one byte checksum (ADDC) calculation.
        bbbb is two byte undefined value.
          cc  is one byte binary count of bytes in numbered statement line.
```

Program file structure : CS/386 mode.

Method invoked by "SELECT NEW" followed by SAVE or RESAVE.
The "text line" content is a combination of textatoms and ascii text, refer to the table "BASIC-2 Verb atoms".

.386 normal mode -- program sectors.

```
60 FileName FD 00 ... 00                                 Header record
00 FF hhhh text line OD 0000                             Program record
   FF hhhh text line OD 0000
   ...
   FF hhhh text line OD 0000 FD

00 FF hhhh text line OD 0000                             Program record
   FF hhhh text line OD 0000
   ...
   FF hhhh text line OD 0000 FD

20 FF hhhh text line OD 0000                             Trailer record
   FF hhhh text line OD 0000
   ...
   FF hhhh text line OD 0000 FE
```

```
.386 wrap   mode.
60 FileName FD 00 ... 00                                 Header record
00 FF aa FFA0 bbbb
       cc hhhh text line statement
       cc hhhh text line statement
00 FF  aa continued text line statement
       cc hhhh text line statement
20 FF  aa continued text line statement
       cc hhhh text line statement
```

```
where     aa  is one byte checksum (ADDC) calculation.
        bbbb is two byte undefined value.
          cc  is one byte binary count of bytes in numbered statement line.
```

New CS/386 format changes:

The new format we have put in the CS/386 is fairly simple:

1) an indicator of the file in the disk catalog tells what the format is:
   index item for the file denoted by : 1040
   this is displayed on program LIST DCT   as TYPE   P'
   refer also to the index catalog item. line

2). New representation for variables and constants. No text atoms have been changed or added.

```
   a) 7C  XX                          A constant less than 256.
   b) 7D  XX XX                       A MAT variable
   c) 7E  XX XX XX XX XX XX XX XX     A constant greater than or equal 256.
   d) 7F  XX XX                       A variable
```

** Be careful, not all 7C/7D/7E/7F encountered are the leading bytes of the new structure; it is context sensitive.

Text atomization: BASIC-2 Verb atoms

80=LIST	81=CLEAR	82=RUN	83=RENUMBER
84=CONTINUE	85=SAVE	86=LIMITS	87=COPY
88=KEYIN	89=DSKIP	8A=AND	8B=OR
8C=XOR	8D=TEMP	8E=DISK	8F=TAPE
90=TRACE	91=LET	92=FIX(	93=DIM
94=CONTINUE	95=STOP	96=END	97=DATA
98=READ	99=INPUT	9A=GOSUB	9B=RETURN
9C=GOTO	9D=NEXT	9E=FOR	9F=IF
A0=PRINT	A1=LOAD	A2=REM	A3=RESTORE
A4=PLOT	A5=SELECT	A6=COM	A7=PRINTUSING
A8=MAT	A9=REWIND	AA=SKIP	AB=BACKSPACE
AC=SCRATCH	AD=MOVE	AE=CONVERT	AF=PLOT
B0= STEP	B1= THEN	B2= TO	B3=BEG
B4=OPEN	B5=CI	B6=R	B7=D
B8=CO	B9=LGT(	BA=OFF	BB=DBACKSPACE
BC=VERIFY	BD=DA	BE=BA	BF=DC
C0=FN	C1=ABS(	C2=SQR(	C3=COS(
C4=EXP(	C5=INT(	C6=LOG(	C7=SIN(
C8=SGN(	C9=RND(	CA=TAN(	CB=ARC
CC=#PI	CD=TAB(	CE=DEFFN	CF=TAN(
D0=SIN(	D1=COS(	D2=HEX(	D3=STR(
D4=ATN(	D5=LEN(	D6=RE	D7=#
D8=%	D9=P	DA=BT	DB=G
DC=VAL(	DD=NUM(	DE=BIN(	DF=POS(
E0=LS=	E1=ALL	E2=PACK	E3=CLOSE
E4=INIT	E5=HEX	E6=UNPACK	E7=BOOL
E8=ADD	E9=ROTATE	EA=$	EB=ERROR
EC=ERR	ED=DAC	EE=DSC	EF=SUB
F0=LINPUT	F1=VER(	F2=ELSE	F3=SPACE
F4=ROUND	F5=AT(	F6=HEXOF(	F7=MAX(
F8=MIN(	F9=MOD(	FA=DATE	FB=TIME

Values FC - FF are reserved.

FC=
FD= used as end of data marker in each program sector on disk.
FE = used as end of file marker in last program sector on disk.
FF = used with hhhh designation to denote line number reference FFhhhh.

An example:

```
   BASIC code     1125  GOTO   2000
   on disk is: 0D 0000 FF11 259C FF20 000D.
   grouped is: 0D 0000  FF1125    9C FF2000 0D.
   meaning     cr thread  1125  GOTO   2000 cr
```

Analysis of sample program using "OLD",ie MVP and "NEW" ie. 386 formats.

```
Program text --
10 REM .
20 PRINTUSING 3500,A$,B$,C$,A1,3*A,45/B*D1
3500 % #####  ##### # $###,###.##  ###,### LAST FIELD
```

"OLD" form disk dump -

```
40 O L D            FD
20FF 0010 A22E 0D00 00FF 0020 A7FF 3500.2C41 242C 4224 2C43 242C 4131 2C33 2A41
2C34 352F 422A 4431 0D00 00FF 3500 D820.2323 2323 2320 2023 2323 2323 2023 2024
2323 232C 2323 232E 2323 2020 2023 2323.2C23 2323 2020 4C41 5354 2046 4945 4C44
0D00 00FE
```

"OLD" form analysis -

```
20   FF 0010 A22E 0D
          10 REM .
0000 FF 0020 A7FF 3500 2C41 242C 4224 2C43 242C 4131 2C33 2A41 2C34 352F 422A
4431 0D
    20 PRINTUSING 3500   ,A    $ ,  B$    ,  C$,    A1,    3   *A    ,45/B*D1
0000 FF 3500 D820.2323 2323 2320 2023 2323 2323 2023 2024 2323 232C 2323 232E
        3500  % .  # #  # #  # .   . #  # #  # #  . #  . $  # #  # ,  # #  # .
        2323 2020 2023 2323.2C23 2323 2020 4C41 5354 2046 4945 4C44 0D00 00FE
         # #  . .  . #  # #  , #  # #  . .  L A  S T  . F  I E  L D
```

"NEW" form disk dump -

```
60  N E W           FD
20FF 0010 A22E 0D00 00FF 0020 A7FF 3500.2C7F F141 2C7F F142 2C7F F143 2C7F 1041
2C7C 032A 7FF0 412C 7C2D 2F7F F042 2A7F.1044 0D00 00FF 3500 D820 2323 2323 2320
2023 2323 2323 2023 2024 2323 232C 2323 232E 2323 2020 2023 2323.2C23 2323 2020
4C41 5354 2046 4945 4C44 0D00 00FE
```

"NEW" form analysis -

```
20   FF 0010 A22E 0D
         10 REM .

00 00FF 0020 A7FF 3500.2C 7FF141 2C 7FF142 2C 7FF143 2C 7F1041
    20 PRINTUSING 3500  ,   A$    ,  B$     ,  C$      ,    A1
2C 7C03 2A 7FF041 2C 7C2D  2F 7FF042 2A 7F1044 OD
,    3   *   A      ,  45    /    B    *    D1

00 00FF 3500 D820 2323 2323 2320 2023 2323 2323 2023 2024 2323 232C 2323 232E
        3500  % .  # #  # #  # .   . #  # #  # #  . #  . $  # #  # ,  # #  # .
        2323 2020 2023 2323.2C23 2323 2020 4C41 5354 2046 4945 4C44 0D00 00FE
         # #  . .  . #  # #  , #  # #  . .  L A  S T  . F  I E  L D
```

THREE BYTE ADDRESSING:

Before using or setting up a Three Byte Address, an address with 65536 sectors or more, it is critical to have a proper understanding of the concept to insure proper use and data integrity. A basic understanding of the 2200 disk file structure and hexadecimal numbers is necessary.

There are currently 3 disk index types that can be used with the DS. The first 2 types are similar and used on all 2200 disk drives. They are sometimes referred to as type 0 and type 1. Type 0 is the original disk index method that has been in effect since the first 2200 disk drive. With Basic-2 Multiuser O/S Release 2.5 a new more efficient disk index structure was made available. The difference between the 2 types was the method in locating the filenames in the disk index. Once setup as type 0 or 1, for many the difference was transparent although in a large indexing scheme the type 1 method could respond faster. The SCRATCH DISK command is used to define the index type:

```
Type 0:   SCRATCH DISK T/Dxx, LS=24, END=65024
Type 1:   SCRATCH DISK ' T/Dxx, LS=24, END=65024
3 Byte:   SCRATCH DISK & T/Dxx, LS=24, END=65024
```

With the LIST command you can always identify the index type. A type 1 index will display a hash mark one position to the right of the index sectors. Three byte displays the &. Following the SCRATCH statements from above, a LIST would display the following:

```
Type 0:                     Type 1:                      3 Byte:

LIST DCT/Dxx                LIST DCT/Dxx                 LIST DCT/Dxx

INDEX SECTORS = 00000024    INDEX SECTORS = 00000024'    INDEX SECTORS = 00000024&
END CAT. AREA = 00065024    END CAT. AREA = 00065024     END CAT. AREA = 00065024
CURRENT END   = 00000023    CURRENT END   = 00000023     CURRENT END   = 00000023
```

Three Byte Addressing is a reference to the number of bytes set aside in the index of a disk for a sector address. With index types 0 and 1 only 2 bytes were set aside. In a 2 byte field the largest value possible is FFFF. In hexadecimal FFFF is equal to 65535 which is why up until now this was the maximum disk address size. Because the first few bytes of sector 0 of a disk are used to hold the index information, the address fields had to be shifted to the right to provide the space needed. The following is a byte by byte breakdown of the first 16 bytes of sector 0 for the 3 index types:

```
Type 0:   SCRATCH DISK T/Dxx, LS=24, END=65024
               byte  0   1     2 3     4 5     6 7 8 9 A B C D E F.....
                    00  18    0018    FE00    00000000000000000000

Type 1:   SCRATCH DISK ' T/Dxx, LS=24, END=65024
               byte  0   1     2 3     4 5     6 7 8 9 A B C D E F.....
                    01  18    0018    FE00    00000000000000000000

3 Byte:   SCRATCH DISK & T/Dxx, LS=24, END=65024
               byte  0   1 2   3 4 5   6 7 8   9 A B C D E F.....
                    02  0018  000018  00FE00  00000000000000
```

MORE

Legend:

```
        Index Type    = Byte 0.  (00 = type 0, 01 = type 1, 02 = 3 byte)
        Index Sectors = Byte 1 for type 0 and 1.      (18 hexadecimal = 24)
                        Byte 1&2 for 3 byte.        (0018 hexadecimal = 24)
        Current End   = Byte 2&3 for type 0 and 1.  (0018 hexadecimal = 24)
                        Byte 3,4&5 for 3 byte.    (000018 hexadecimal = 24)
        End Cat. Area = Byte 4&5 for type 0 and 1.  (FE00 hexadecimal = 65024)
                        Byte 6,7&8 for 3 byte.    (00FE00 hexadecimal = 65024)
        Undefined     = Bytes 6-F for type 0 and 1.
                        Bytes 9-F for 3 byte.
```

With a 3 byte index maximum 'Index Size' is increased from 256 to 65536.
    Type 0&1 (1 byte) FF = 256 (includes 0) 3 byte (2 bytes) FFFF = 65536
Maximum 'End Catalog Area' is increased from 65535 to 16,777,215.
    Type 0&1 (2 bytes) FFFF = 65535   3 byte (3 bytes) FFFFFF = 16777215

A similar byte displacement is used with each filename entry used to locate the 'START' and 'END' fields displayed with the LIST command. In all 3 index types, each filename entry is allotted 16 bytes. The following example shows the byte displacement between the type 0/1 and 3 byte filename entries:

```
    Type 0/1: NAME      TYPE  START     END       USED      FREE
              FILENAME   P    00000024  00001000  00000960  00000007
                   byte  0    1   2 3     4 5      6 7     8 9 A B C D E F.....
                         10   80  0018    03E8    0000   46494C454E414D45

    3 Byte:   NAME      TYPE  START     END       USED      FREE
              FILENAME   P    00000024  00001000  00000960  00000007
                   byte  0    1   2 3 4   5 6 7            8 9 A B C D E F.....
                         10   80  000018  0003E8           46494C454E414D45
```

Legend:

```
        File Status = Byte 0 for all 3 types.    00 = unassigned.
                                                 10 = active.
                                                 11 = scratched.
                                                 21 = re-used scratch entry.
        File Type   = Byte 1 for all 3 types.    00 = data file.
                                                 80 = program file (old format)
                                                 40 = program file (386 format)
        START Address = Byte 2&3 for type 0/1.      (0018 hexadecimal = 24)
                        Byte 2,3&4 for 3 byte.    (000018 hexadecimal = 24)
        END Address   = Byte 4&5 for type 0/1.      (03E8 hexadecimal = 1000)
                        Byte 5,6&7 for 3 byte.    (0003E8 hexadecimal = 1000)
        Undefined     = Bytes 6&7 for type 0/1.
        NAME of File  = Byte 8-F for all 3 types.
                                  46494C454E414D45 in ASCII = FILENAME.
```

WARNING: Changing the index structure may create problems with some programs possibly resulting in data integrity errors. Certain BASIC-2 programs directly utilize specific bytes of the disk index. For those programs that do, changes would likely be needed to insure proper operation. Before changing the index type, a careful analysis of the programs to be used at that address should be made to determine if any direct utilization of the index bytes is being used. Additionally, if any BIN or VAL commands are used to

MORE

manipulate this indexing data, they would need to be expanded in size for 3 byte addressing for proper calculations. BIN and VAL work from left to right and would therefore ignore the low order byte if currently set for 2 bytes. Proper use of a new command, SELECT 3 ON/OFF, is also critical to error-free operation. Please read the next section on SELECT 3 ON/OFF for details. Backups should always be done before any changes are made.

SELECT 3 ON/OFF Command:

Whenever 3 byte addressing is to be used, a new SELECT command, SELECT 3 ON must be executed. When using a standard type 0 or type 1 index, SELECT 3 should be OFF. SELECT 3 ON allows certain instructions such as DATA LOAD BA to work with addresses greater than 65535. Otherwise an error such as P34, illegal value, would occur. SELECT 3 must be turned ON and OFF for each partition. The proper syntax to turn 3 byte addressing ON or OFF is as follows:

```
SELECT 3 ON

SELECT 3 OFF
```

To determine if '3 byte' addressing is turned on for a particular partition, use the LIST SELECT command from that partition. It is used to show the status of all the available selectable options.

Known Anomalies with 3 Byte Addressing with Turbo O/S 1.1:

1. Filenames may be assigned to sector 0 of the index. Sector 0 should have no filename entries. (This is subject to change.)

2. The RENAME command in some instances could corrupt the disk index.

3. The LOAD DA command will not load a program from a sector address beyond 65535.

4. Cannot boot the system from a 3 byte address if the CPU boot files are out beyond sector 65535.

Note: The following 2200 utility programs do not support 3 Byte Addressing:

MENU pick	PROGRAM
Move File	@MOVEFIL
Backup Disk Platters to Disk Drive	@BACKUP
Restore Disk Platters from Disk Drive	@RECOVER
System Install	@INSTALL
Moving a Selected List of Files	@TO.CREF

END