

Tuning the VSAM KSDS Index CISZ

November 22, 2004

By: Eugene S. Hudders

CICS/TS Performance Tuning Using C\TREK Course

This seminar covers a lot of tuning material and will require 8 hours a day. The total time for the seminar is 40 hours.

The followings topics included in this course are:

- Introduction to CICS Performance Tuning
- Using Operating System Information to Tune CICS/TS
- Tuning CICS/TS Processor Cycles
- Tuning Real Storage in CICS/TS
- Tuning Virtual Storage in CICS/TS
- Tuning CICS/TS Transaction Controls
- Tuning On Line VSAM Files
- Tuning NSR Files in CICS/TS
- Tuning CICS/TS LSR Buffer
- Tuning the Index CISZ
- Reviewing VSAM Free Space
- Reviewing the DB2 Interface

If you wish to attend this course, please send us an email at ctrek@actpr.com

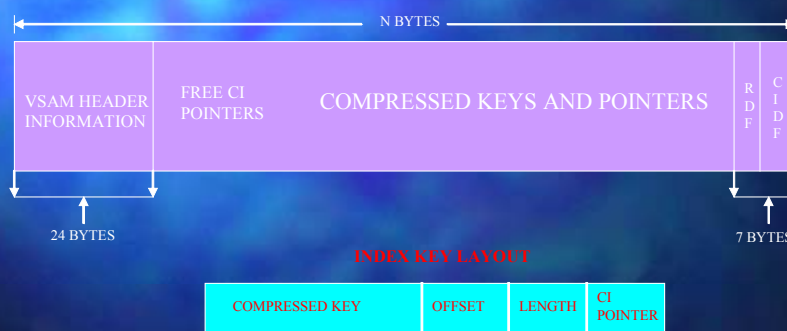
A VSAM KSDS tuning area that receives little attention is the index CISZ. In many cases the VSAM default algorithms determine the index CISZ associated with the KSDS cluster. The default index selected by VSAM is generally acceptable especially after the change that occurred in the selection criteria in z/OS V1.R3. Tuning the index CISZ is an important issue for two reasons. The first reason is the number of index levels associated with the file and the second reason is avoiding key compression problems. This article reviews the selection of index CISZ including an overview of the index record structure within the CI.

Let us first review the general structure of the index CI. There are two types of index CIs. The first index CI is the lowest level index called the Sequence Set Index (SSI). There is one SSI for every data CA in the file. This index record has to have sufficient space to accommodate the compressed high key of every data CI in the data CA. In other words, if there are 180 CIs/CA, then the SSI has to be sufficiently large to accommodate 180 high keys plus the control information. The second type of index CI is for the second and higher levels within a file and is called the Index Set (IS). The second level IS records contains compressed high keys from each of the SSI records in its scope. Once the IS record is full, another IS record at the same level is created. The KSDS index structure requires that there be one high level index record for each file. Thus, whenever the second record at the same level is created, then a higher-level index record is created to become the high-level index record. The Relative Byte Address (RBA) of this index record is maintained in the VSAM catalog and is printed out in the index portion of a LISTCAT. An SSI can be the high level index record for properly defined small files that are one cylinder or less in size.

Figure 1 provides a general layout of an index record and the control information for a compressed key.

DEFINING THE INDEX CI SIZE

■ GENERAL INDEX RECORD LAYOUT



Copyright C\TREK Corporation 2003

12

Figure 1. General Index Record Layout

The index CI contains control information in the first 24 bytes and has a seven byte VSAM RDF/CIDF at the end. Therefore, you lose 31 bytes out of every index CI for control data. The compressed keys and associated key control information have to fit in the remaining space. Each key is compressed from the front and the back. The front key compression is easy because it is a matter of setting up a root key and then every key after that one gets the duplicate digits truncated. In other words, equal value characters are truncated from the front of the key. Back key compression is a little harder to visualize but it is simply the truncation of non-significant bytes from the end of the key. When re-generating the key, simply copy the eliminated characters from the root key to the front portion of the compressed key and fill out the end of the remaining bytes with X'FF' to complete the length of the key. This X'FF' filler indicates that you could add a new key with a higher value than the highest key you previously loaded into the CA but is lower than the next key in the next CA. In some cases, the VSAM compression algorithm can compress a key to zero bytes.

Compressed keys have a control field placed at the end of each compressed key. It is variable in length but it is usually 3 bytes for the SSI records. The control structure is comprised of one byte to indicate where the relative position of the compressed key is to be placed in order to locate how many characters have to be obtained from the root key, one byte to indicate the length of the compressed key and one to three bytes to be used as a relative CI pointer within the CA (Figure 1). If the file contains less than 256 CI/CA, then a one-byte field will do. This is

where the usual three bytes of control information are generated for the sequence set records (SSI). If you want to make a quick estimate of how many keys will fit into a selected CISZ use the following formula:

$$\# \text{ Of Keys in CI} = (\text{Index CISZ}-31)/((\text{Key Length}/3)+3)$$

The estimated key compression ratio is around 1/3. If the selected CISZ for the SSI is not sufficiently large enough to contain all of the compressed keys in a CA, VSAM will simply mark all the remaining CIs as free CIs and move on to the next CA to continue loading data. These extra CIs are above what you may have reserved with the CA FSPC percentage. **No message is sent indicating that this condition is occurring.** This condition is called a key compression problem. The only way that you may find out you have this condition (outside of having a performance tool or routine) is that your file takes up more space than you had computed or goes into early extents. Another possible identification is that the file gets CA splits almost immediately with the first insertions even though it reflects a lot of free space available. So, choosing an appropriate index CISZ can affect the space you allocate for the file. Potential candidates for this type of occurrence are large files with long keys (greater than 40 bytes) that use large data CISZes (e.g., 18K), which result in small default index sizes such as 512 bytes. VSE has been good in controlling this problem over the years because it used the 1/3 ratio in computing the size. MVS on the other hand thought that it could compress all keys to 8 bytes and thus key compression was more prevalent in this operating system. This problem was fixed in z/OS V1.R3.

If VSAM had to uncompress all the keys in order to find the desired key, then a lot of CPU time would be used during the key searches. So, VSAM divides the index record into sections in order to simplify and accelerate the search. Each section has a "root" key. There is a pointer in the header that points to first section to work with the root key. This section entry has a section length associated with it that VSAM uses to compute where the next section is located. So, when VSAM looks for a key, it does so by section. When it finds the approximate key location, it then begins to uncompress the appropriate keys in that section. The number of sections in an index CI is the square root of the number of data CIs/CA. The section control is two bytes in length. So, when you compute the average compressed key size using the previous formula, you will not be including the section overhead. You may want to include the section bytes in the computation. However, the information in the section control fields is not sufficiently large to significantly affect the estimated key size computed in the formula. So, don't worry about this difference. However, if you want to include the effect section length on the computed average key size, simply multiply the result from the computed average size by 103 to 105%, depending on how conservative you want the result to be.

The twenty-four (24) bytes at the beginning of the index CI is used to control the information used by VSAM. Some of the important fields are as follows:

- +X'02' – the length of the key control information (1 byte)
- +X'03'—the length of the vertical pointer (1 byte)
 - X'01' – 1 byte vertical pointer
 - X'03' – 2 byte vertical pointer
 - X'07' – 3 byte vertical pointer
- +X'04' – the pointer to the data CA (SSI) or index record (IS) (4 bytes)
- +X'08' – the horizontal pointer to next same level index record (4 bytes)
- +X'10' – the index level (1 byte)

The above information is provided to provide a flavor of the control data contained in the index record. **Figures 2 and 3** provide the beginning and ending of an SSI record that can be used to review the information in the index record. The first thing that should be determined is the type of index record (SSI or IS) being evaluated. This can be determined by looking at +X'10' that provides the index level. In this case (**Figure 2**) we have a value of X'01' that indicates that we are working with an SSI record or first level index. The value of X'03' at offset +X'02' indicates that the key control information is three bytes long. This is followed by a value of X'01' at offset +X'03' that indicates that the vertical pointers to the data CIs is one byte long. The vertical pointers to the data CIs are simply the relative sequence number of the data CI within the CA starting with X'00'. A one-byte vertical pointer is valid for up to 256 CIs/CA (X'00' to X'FF').

```

DHEX ADVANCED COMPUTER TECHNOL  C\TREK ON-LINE      21E2D000  DATE 11/23/2004
APPLID CICSTS22                   MEMORY DISPLAY      TIME 18:50:09
VERSION 6.2                       TERM 0216

21E2D000 00000000 07F90301 00000000 00000800 00000000 .9.....
21E2D010 00000010 01000018 074007D7 00000000 00000000 .....P.....
21E2D020 00000020 00000000 00000000 00000000 00000000 .....
21E2D030 00000030 00000000 00000000 00000000 00000000 .....
21E2D040 00000040 00000000 00000000 00000000 00000000 .....
21E2D050 00000050 00000000 00000000 00000000 00000000 .....
21E2D060 00000060 00000000 00000000 00000000 00000000 .....
21E2D070 00000070 00000000 00000000 00000000 00000000 .....
21E2D080 00000080 00000000 00000000 00000000 00000000 .....
21E2D090 00000090 00000000 00000000 00000000 00000000 .....
21E2D0A0 000000A0 00000000 00000000 00000000 00000000 .....
21E2D0B0 000000B0 00000000 00000000 00000000 00000000 .....
21E2D0C0 000000C0 00000000 00000000 00000000 00000000 .....
21E2D0D0 000000D0 00000000 00000000 00000000 00000000 .....
21E2D0E0 000000E0 00000000 00000000 00000000 00000000 .....
21E2D0F0 000000F0 00000000 00000000 00000000 00000000 .....

ENTER REFRESH PF1 HELP PF2 MEMORY SEARCH PF3 PREV PAGE PF4 LAST 10 ADDRESSES
PF7 BACKWARD PF8 FORWARD PF9 FIRST PF10 SWAP CLR MAIN MENU

```

Figure 2. VSAM KSDS Index – (Part 1)

```

DHEX ADVANCED COMPUTER TECHNOL  C\TREK ON-LINE  21E2D700  DATE 11/23/2004
APPLID CICSTS22                  MEMORY DISPLAY  TIME 18:50:33
VERSION 6.2                      TERM 0216

21E2D700 00000700 00000000 00000000 00000000 00000000 .....
21E2D710 00000710 00000000 00000000 00000000 00000000 .....
21E2D720 00000720 00000000 00000000 00000000 00000000 .....
21E2D730 00000730 00000000 00000000 000000F1 F2F2F1F5 .....12215
21E2D740 00000740 0D0515F1 F0F8F1F8 7AF2F00D 08140013 ...10818:20....
21E2D750 00000750 F2F9F00E 0313F313 0112F6F1 F87AF1F6 290...3...618:16
21E2D760 00000760 7AF00F08 11F2F1F1 F50E0410 0021F1F0 :0...2115....10
21E2D770 00000770 F1F4F1F3 0C060FF2 F00E020E F87AF4F0 1413...20...8:40
21E2D780 00000780 11040DF6 F1F60F03 0C001CF4 F1F27AF3 ...616....412:3
21E2D790 00000790 0F050BF2 7AF21103 0AF5F27A F0F14BF1 ...2:2...52:01.1
21E2D7A0 000007A0 130709F4 F1F17AF1 0F050800 20F9F0F3 ...411:1....903
21E2D7B0 000007B0 0D0307F3 F1F10F03 06F9F0F1 0D0305F3 ...311...901...3
21E2D7C0 000007C0 F0F10E03 040027C3 C9C3E2E3 E2F2F2F2 01....CICSTS222
21E2D7D0 000007D0 F0F0F4F0 F8F2F600 1003F2F5 0E020213 0040826...25....
21E2D7E0 000007E0 0001C3C9 C3E2E3E2 F2F2F2F0 F0F4F0F8 ..CICSTS22200408
21E2D7F0 000007F0 F1F4F1F3 7AF20014 000007F9 07F90000 1413:2....9.9..

ENTER REFRESH PF1 HELP PF2 MEMORY SEARCH PF3 PREV PAGE PF4 LAST 10 ADDRESSES
PF7 BACKWARD PF8 FORWARD PF9 FIRST PF10 SWAP CLR MAIN MENU

```

Figure 3. VSAM KSDS Index – (Part 2)

As we are working with an SSI index record, the RBA address of the data CA within the file can be found at an offset of +X'08'. In this particular case, the RBA contains X'00000000' that indicates that this is the first CA of the file. This figure will increase by a CA size for each subsequent SSI record immediately after the file load is complete. The CA size is nothing more than product of the number of data CIs/CA times the data CISZ. To locate a particular data CI you would use the RBA figure where the CA begins and add to this value the multiplication of the vertical pointer contents times the data CISZ. This multiplication gives you relative location of the data CI within the CA. VSAM uses this RBA value to compute the physical disk address (MBBCCCHR) address of the CI. It should be noted that for VSAM/E files, the value in the index record pointer is the relative CA number and not an RBA. So, VSAM has to multiply this value by the CA size to compute the actual RBA that is eight bytes long for VSAM/E files.

The final field we will discuss is the horizontal pointer field. This field is used to maintain the logical sequence of the file, especially after CA splits, by pointing to the next logical sequential index record. In the case of the SSIs, this field is used to process the file sequentially by following the chain. CA splits cause a current CA to be divided into two CAs with the second or new CA being added at the end of the data High Used RBA (HURBA) that is located at the end of the file. This CA could be placed in a different extent and/or volume (if available). The new SSI record that controls the split CA is also added after the last index record in the file's index component. To maintain the logical sequence for sequential processing, the horizontal pointer of the original splitting CA would point to the new SSI at the end of the index file. This new SSI would point back to the next logical SSI to maintain the logical sequence of the data records in the file.

While on the topic of splits, we should note that both CI and CA “splits” could occur for the index portion of the file. These “splits” are not as dramatic as data CI/CA splits with regards to performance but they do add additional overhead. Remember that the second level IS record accommodates a series of high keys from the SSI records. Once the IS record gets filled to a certain point, another IS record of the same level is added and the process continues. When you get a data CA split, a new SSI key has to be added to the IS record. If the new key does not fit, then you must split this IS record to make room for the new key. This means that you need to add this new IS record at the end of the index component. The horizontal pointers are updated to maintain logical sequence of the IS records. There is nothing that can be done to prevent index splits as the free space parameter only applies to the data portion of the file. An index CI split refers to an SSI record split while an index CA split refers to IS record split.

The final index control information is the RDF/CIDF fields located at the end of the index record (Figure 3). The information is as follows: X'0007F907F90000'. This simply indicates that there is only one record within the index with a length of X'07F9' (RDF information) and that the free space begins at a displacement of +X'07F9' and has zero bytes available (CIDF information). VSAM handles the data within the index record. Finding the RDF/CIDF is important because the first compressed key, the one that refers to the first CI in the CA (lowest key within the CA) is placed at the end of the index record, just before the RDF/CIDF information. The compressed keys are stored from the high end of the record back to the beginning of the record. This is contrary to what is shown in many VSAM books and articles.

Let us use Figure 4 for the discussion regarding the compressed key structure. This particular file has a key length of eight bytes and the control information is three bytes long. The compressed key starts just to the left of the RDF field and contains X'C1D7E2D4F0F6000600'. The key control information (last three bytes-X'000600') indicates that the first byte of the compressed key goes into position X'00', that the compressed key is six bytes long and that this key belongs to relative CI X'00'. The compressed key is X'C1D7E2D4F0F6'. This is a root key because the first byte (X'C1') goes into the first position (X'00') when re-structuring the key. As the key had a total of eight bytes, then there was no front key compression and two bytes were compressed from the end of the key. So, when this key is re-expanded to eight bytes it will look like: X'C1D7E2D4F0F6FFFF'.

```

DHEX ADVANCED COMPUTER TECHNOL  C\TREK ON-LINE      22469700  DATE 11/26/2004
APPLID CICSTS22                   MEMORY DISPLAY      TIME 10:31:13
VERSION 6.2                       TERM 0216

22469700 00004700 00000000 000000D7 D7C9F1F1 000516D7 .....PPI11...P
22469710 00004710 C2E2E3F1 F2000615 0013D6C6 D3F1F400 BST12....OFL14.
22469720 00004720 0519D5D8 D7F0F7F0 F1000714 D4D5F1F8 ..NQP0701...MN18
22469730 00004730 000413E2 C3E3F1F3 01051200 24D3C5F0 ...SCT13....LE0
22469740 00004740 F7F0F100 0611C4F0 F8020310 D3C3C1E3 701...D08...LCAT
22469750 00004750 F1F4F400 070F001A D2E5D4C8 E400050E 144....KVMHU...
22469760 00004760 D9E3F204 030DE5D4 C8C1E3C5 010617D2 RT2...VMHATE...K
22469770 00004770 C500020C 001ED1C3 E3F2F000 050BC7E3 E....JCT20...GT
22469780 00004780 C1E7F0F8 00060AE5 D6D3F001 0409001A AX08...VOL0....
22469790 00004790 C6C9D6F0 F8000508 C6C3E3C5 000407C5 FIO08...FCTE...E
224697A0 000047A0 D9D9F0F5 F0000606 001DC4E3 C1C4F1F5 RR050....DTAD15
224697B0 000047B0 F0F10008 05D4D7F1 F4010404 C4C4E2F1 01...MP14...DDS1
224697C0 000047C0 F1F00006 03001CC3 D7E7F0F7 F0F20007 10....CPX0702..
224697D0 000047D0 18C90101 02C5C5C3 F0F60105 1AC3C2C1 .I...EEC06...CBA
224697E0 000047E0 F0F6F000 0601E2C3 D3F0F6F0 F501071B 060...SCL0605...
224697F0 000047F0 C1D7E2D4 F0F60006 000047F9 47F90000 APSM06....9.9..

ENTER REFRESH PF1 HELP PF2 MEMORY SEARCH PF3 PREV PAGE PF4 LAST 10 ADDRESSES
PF7 BACKWARD PF8 FORWARD PF9 FIRST PF10 SWAP CLR MAIN MENU

```

Figure 4. VSAM KSDS Compressed Keys Structure

The next key in this section has a value of X'E2C3D3F0F6F0F501071B'. The control information contains X'01071B' meaning that the compressed key is seven bytes long (07) and when it is re-expanded, the first byte of the key will occupy the second position of the key (01). The first byte is retrieved from the root key which is the value of X'C1' (see previous key). The key received a one- byte key compression from the front only. So, the fully expanded key would look like: X'**C1**E2C3D3F0F6F0F5'. Another interesting thing can be observed from the control field that is that this key corresponds to relative data CI X'1B' and not X'01' that should have been the next CI. This reflects that CI number X'00' suffered a CI split and that CI number X'1B' was used to accommodate the split key. The contents of the next compressed key confirm the split. The contents look like: X'C3C2C1F0F6F0000601'. Note that the compression rate has not been close to the 1/3 previously mentioned. Small sizes keys (e.g., eight bytes) may not compress well.

The final area regarding the index layout that we will discuss is the free space information maintained within the SSI record (Figure 5). If CA free space is specified or unused CIs result as a result of key compression problems, an inventory of available CIs is kept within the SSI record. The list of free CIs available to handle CI splits within the CA are kept as a list at offset +X'18' in the SSI record. Normally, the list may be in declining relative CI number (Hex) until an X'00' is found. However, if all the records from a CI are deleted, this CI can be reclaimed as free space and break the declining sequence. In this example we can see that CIs starting at X'2C' through X'1C' are available to handle CI splits within this CA.

```

DHEX ADVANCED COMPUTER TECHNOL  C\TREK ON-LINE  22465000  DATE 11/23/2004
APPLID CICSTS22  MEMORY DISPLAY  TIME 18:59:11
VERSION 6.2  TERM 0216

22465000  00000000  47F90301  00000000  0000D800  00000000  .9.....Q.....
22465010  00000010  01000029  470C47CE  2C2B2A29  28272625  .....
22465020  00000020  24232221  201F1E1D  1C000000  00000000  .....
22465030  00000030  00000000  00000000  00000000  00000000  .....
22465040  00000040  00000000  00000000  00000000  00000000  .....
22465050  00000050  00000000  00000000  00000000  00000000  .....
22465060  00000060  00000000  00000000  00000000  00000000  .....
22465070  00000070  00000000  00000000  00000000  00000000  .....
22465080  00000080  00000000  00000000  00000000  00000000  .....
22465090  00000090  00000000  00000000  00000000  00000000  .....
224650A0  000000A0  00000000  00000000  00000000  00000000  .....
224650B0  000000B0  00000000  00000000  00000000  00000000  .....
224650C0  000000C0  00000000  00000000  00000000  00000000  .....
224650D0  000000D0  00000000  00000000  00000000  00000000  .....
224650E0  000000E0  00000000  00000000  00000000  00000000  .....
224650F0  000000F0  00000000  00000000  00000000  00000000  .....

ENTER REFRESH PF1 HELP PF2 MEMORY SEARCH PF3 PREV PAGE PF4 LAST 10 ADDRESSES
PF7 BACKWARD PF8 FORWARD PF9 FIRST PF10 SWAP CLR MAIN MENU

```

Figure 5. VSAM KSDS Index with Free CIs

Key compression problems result in a file occupying more space than required. This wasted disk space can result in a significant savings when the key compression problem is corrected. As previously mentioned, there is no VSAM message to alert you that this condition exists. So, you will have to use other means to determine if the problem exists such as extra extents or CA splits occurring almost immediately in a file with a lot of free space. Another method to determine if a key compression problems exists is to print out the index file immediately after loading and take a look at the space in the SSI records (X'01'at +X'10') immediately following the index control (+X'18') and the first byte of the index data. The CCHH (physical disk address) where the index is placed on the disk and the volume number can be obtained from the file's LISTCAT. If there are a lot of binary zeroes (low-values) immediately following the free CI list starting at +X'18' and the first compressed key, then you probably do not have a key compression problem.

The simple solution to fixing key compression is simply to increase the index CISZ to a higher multiple. This will require a re-load of the file after a delete and define of the cluster. You cannot change the CISZ via an alter command. You can use the following formula to determine the size of the SSI record required to handle the particular key length:

$$\text{\# Of Bytes Required} = (((\text{Key Length}/3) + 3) * \text{\# Of CIs/CA}) * 1.05$$

Round the result to the next CISZ multiple, that is if the result was 825 bytes, then the appropriate CISZ would be 1024. There may be other considerations when performing this rounding such as if the file is being accessed using LSR and CICS. If the formula yields an index size that requires an index CISZ of 1536 bytes, you may want to increase the size to 2048 (2K) to avoid virtual storage fragmentation because there is no 1536 buffer when using LSR in CICS. Another reason why you may want to increase the index CISZ may include the control of index levels.

Controlling the number of index levels is not done in many installations. This is an important tuning item that is often overlooked; mainly because the sheer number of VSAM files that have to be analyzed in an installation. VSAM KSDS files should be limited to a maximum of three (3) index levels. Lowering a file that has 4 index levels to three results in a reduction of 20% of CPU cycles when all the records are in the LSR buffers. To locate a record in a file that has 4 index levels you would have to search the LSR pool four times for the index CIs and once for the data CI. So, lowering the index levels to 3 would result in one less search. In addition, lowering the number of index levels results in less IS records that have to be buffered. Simply increase the index CISZ to lower the number of index levels. Increasing the CISZ will probably have no additional value for the SSI records but will increase the number of SSI keys that can be handled by the IS record, resulting in a lower number of IS records needed to handle the file. This also results in having fewer buffers to allocate for the file.

In some cases, you may be able to reduce a three level file to two levels by simply increasing the index CISZ. For example, imagine that a 1.5K SSI CI can hold up to 180 high keys, then the IS CI should be able to hold between 150 to 180 high keys depending on the compression ratio and that the compressed key control information is larger for the IS CI. Therefore, if the CA is one cylinder in size, then the IS record would be able to accommodate a file between 150 to 180 cylinders before another second level IS record is required that would force a third level. By simply increasing the index CI size to 2K may be sufficient for the file to accommodate the entire file in one IS record and only have two levels. This can be considered tweaking the file and could be considered overkill but can result in CPU savings for an active file. Determining the number of levels manually can be tedious and would require a mechanized tool.

Another area that requires attention when selecting the index CISZ is how the index CISZ is requested. Of particular importance is when the CISZ for a VSAM KSDS is requested by specifying the CISZ at the cluster level only. In this case, the specified index CISZ applies to both the data and index. This could result in an excessively large index CISZ. Although specifying the CISZ at the cluster level is viable, users should always specify the data CISZ in the data component and the index CISZ in the index component and avoid specifying any CISZ in the cluster level.

This article covered the importance of selecting the appropriate index CISZ to avoid key compression problems that result in wasted disk space and to control the number of index levels

associated with the file that can reduce the amount of CPU time spent searching indices and number of virtual storage LSR/NSR buffers required to hold the IS records. VSAM's new default CISZ selection algorithm (after z/OS V1.R3) is oriented to reducing key compression problems but not index levels.

If you have any questions, comments or request for more information, please contact us:

Address: C\TREK Corporation PO BOX 560069 MONTVERDE FL 34756	Phone: (407) 469-3600 C\TREK Corporation (787) 756-5620 Advanced Computer Technology
E-mail: ctrek@actpr.com	Fax: (787) 756-5150
Website: www.ctrekcorp.com	Support:: (787) 397-4150 (321) 297-5838 (787) 462-0406