# Table of Contents

1. **FAIRCOM UPDATE GUIDE V7.11** ........................................................................................................... 1  
   1.1 Huge and Segmented File Support ................................................................................................ 2  
   1.2 Unicode Support and Extended Key Segments ............................................................................. 2  
   1.3 Advanced File Encryption ...................................................................................................... ......... 2  
   1.4 Portability Enhancements ..................................................................................................... ......... 3  
   1.5 Enhanced Variable-Length Space Management ........................................................................... 3  
   1.6 Server and c-tree Plus Enhancements ........................................................................................... 3  
   1.7 c-tree Driver Enhancements .................................................................................................... ...... 4  
   1.8 New Function Reference ........................................................................................................ ....... 4  
   1.9 Compatibility Notes .......................................................................................................... ............ 5  
2. **HUGE FILE SUPPORT** ...................................................................................................................... 11  
   2.1 Huge File Basics .............................................................................................................. ........... 11  
   2.2 Xtd8 File Creation Functions ................................................................................................... ... 12  
   2.3 Huge File Creation Example ..................................................................................................... .. 13  
   2.4 Record Offsets Under Huge File Support ................................................................................... 14  
3. **SEGMENTED FILE SUPPORT** ........................................................................................................ 15  
   3.1 Segmented File Basics ......................................................................................................... ....... 15  
   3.2 Automatic Segments ........................................................................................................... ........ 16  
   3.3 SEGMDEF Structure ............................................................................................................ ...... 16  
   3.4 SetFileSegments Function ..................................................................................................... ...... 17  
   3.5 File Segment Example .......................................................................................................... ...... 18  
   3.6 Segmented Dynamic Dump ........................................................................................................ 19  
   3.7 Segmented Dynamic Dump Example ......................................................................................... 19  
4. **EXTENDED FILE CREATION BLOCK STRUCTURE** ................................................................. 21  
   4.1 Extended File Modes (x8mode) ................................................................................................... 21  
5. **UNICODE SUPPORT** ...................................................................................................................... 25  
   5.1 Unicode Concepts ............................................................................................................... .......... 25  
   5.2 Preparation ......................................................................................................................... ................. 25  
   5.3 Storing Unicode Data ......................................................................................................... ........ 26  
   5.4 Unicode Key Segments ......................................................................................................... ..... 27  
   5.5 Unicode File Names .............................................................................................................. ........ 43  
6. **ADVANCED FILE ENCRYPTION** ................................................................................................. 45  
   6.1 Using Advanced File Encryption ............................................................................................ 45  
   6.2 Cryptographic techniques ...................................................................................................... ....... 47  
7. **PORTABILITY ENHANCEMENTS** ............................................................................................. 53  
   7.1 New Platform Support ............................................................................................................. 53
# Table of Contents

7.2     Enhanced Platform Support ................................................................. 54

8.     IMPROVED DELETED SPACE MANAGEMENT ................................. 61
  8.1     Improved Deleted Space Management ........................................ 61
  8.2     Example .................................................................................... 62

9.     SERVER ENHANCEMENTS .............................................................. 63
  9.1     Multi-CPU Performance ................................................................. 63
  9.2     c-tree Server Detects Dropped Clients ........................................ 63
  9.3     Server Detects Inactive Clients ................................................... 64
  9.4     Superfile Space Reclamation ......................................................... 64
  9.5     Server Files Transaction Dependent ........................................... 65
  9.6     File-specific Encryption Control .................................................. 65
  9.7     Diagnostic Enhancements ............................................................ 65
  9.8     Logon Timeout Enhanced ............................................................. 68
  9.9     Change log and dynamic dump buffer sizes .............................. 68
  9.10    Coordinate deletes and dynamic dumps ................................. 68
  9.11    Client-side Broadcast Functionality Enhanced ...................... 68
  9.12    Support PAGE_SIZE Up To 65536 ........................................…. 69
  9.13    CHECKPOINT_INTERVAL adjustments ..................................... 69
  9.14    SystemLog additions ................................................................. 69

10.    C-TREE PLUS ENHANCEMENTS .................................................. 71
  10.1    Transaction Dependent Creates and Deletes ............................ 71
  10.2    Index Any Field Type in a Variable-Length Record .................. 72
  10.3    Adaptive Index Node Splits ....................................................... 73
  10.4    Disk Full Enhancements .............................................................. 74
  10.5    API Enhancements ................................................................. 75
  10.6    Utility Enhancements ................................................................. 77

11.    C-TREE DRIVER ENHANCEMENTS ........................................... 79
  11.1    Performance Enhancements ......................................................... 79
  11.2    New Features ............................................................................ 79
  11.3    Fixes ........................................................................................ 83

12.    NEW FUNCTION DESCRIPTIONS .................................................. 89

13.    FIXES ......................................................................................... 117
  13.1    Critical Issues ........................................................................ 117
  13.2    Serious Issues .......................................................................... 118
  13.3    Other Fixes .............................................................................. 120
**c-tree Plus / c-tree Server CORE database technology**

- **Our Secret is in the Details**

The product literature that accompanies this c-tree Plus V7.11 upgrade communicates FairCom's direction related to the importance of higher level programming interfaces such as SQL. Although a significant focus has been placed on the interface technologies, the key to our success (and your product satisfaction) remains within our CORE technology. This CORE is our strongest asset and the key to the dependability, flexibility, and performance expected from FairCom.

The release of our V7.11 CORE technology introduces a new era for FairCom. By taking the time to step back, review our core, and apply our experience and expertise toward a refreshed architecture, we have been able to take a major step forward with our database technology. This CORE is now the basis for all future FairCom efforts and offers a significant framework for all next generation projects. Issues such as 64-bit and HUGE file support, as well as significant enhancements within the core, make V7.11 an important plateau.

With most software updates, there are a large number of details that go into a new release. Many issues are internal and require no external documentation, while others offer new capabilities. c-tree Plus V7.11 is no exception. In fact, a significant number of internal details, which outweigh the external issues, have been addressed in several subsystems in this release. The graphic displayed on the next page attempts to illustrate the depth and breath of the details that have been addressed within the c-tree Plus and c-tree Server code. Subsequent chapters in this Update Guide document the many features and improvements found in this release.

As always, FairCom thanks you for your business and support of our technology. We appreciate you as a customer, and with your support we will continue to strive to bring you the best database technology.

Thanks,

The FairCom Team
1. FairCom Update Guide V7.11

FairCom is pleased to deliver the results of an extended development effort in this latest product release. This update includes the following product versions:

- **c-tree Plus** File Handler V7.11 – including HUGE file support.
- **c-tree Server** V7.11.56 – with OVER 30 Supported Platforms
- **c-tree Server** V7.11 Software Development Kit (see Chapter 14 of the c-tree
  Plus Programmer’s Reference Guide and Chapter 6 of this guide.)
- **c-tree Drivers** V3.11 – Including ODBC Drivers for c-tree Plus and c-tree V4 and a native Crystal Reports™ Driver.

This Guide is made up of the following sections:

Chapter 1. Overview – An overview of the new FairCom releases
Chapter 2. Huge File Support – 8 byte addressing
Chapter 3. Segmented File Support – Spreading a logical file over multiple physical files
Chapter 4. Extended Creation Block Structure – The key to many features and enhancements added to V7.11
Chapter 5. Unicode – Using Unicode with c-tree Plus
Chapter 6. Advanced File Encryption – c-tree Server SDK enhancements allowing enhanced security
Chapter 8. Variable-Length Space Management – Enhanced space management with the \texttt{RECBYT} segment mode
Chapter 10. c-tree Plus Enhancements – Other features and enhancements added to c-tree Plus.
Chapter 11. c-tree Driver Enhancements – New features and enhancements for the c-tree Plus ODBC Driver, the c-tree V4 ODBC Driver, and the c-tree Crystal Reports Driver.
Chapter 12. Function Reference – Description pages for the new functions implemented in this release
Chapter 13. Fixes – Critical Issues addressed since the previous commercial release.

FairCom has been working hard and we are proud to bring these new features and enhancements to you. Thank you for your commitment to FairCom technology and we hope you find great value within this release. As always, if you should have any questions, or special needs not addressed in this release, we encourage you to contact your nearest FairCom office.

NOTE: Throughout this document the product **c-tree Server** is the new name for the former **FairCom Server**. The product name has been changed beginning with V7.
1.1 Huge and Segmented File Support

**c-tree Plus** Professional Edition greatly improves file size and volume management with support for Huge Files (8 byte file addresses) and Segmented files.

- 8 byte file addresses permit data and index files to grow beyond the 4 GB limit inherent in previous releases. An 8 byte file address supports files on the order of $10^{19}$ bytes or 18,000,000 terabytes. This option allows a great deal of data to reside in a given file. See Chapter 2 for more details on Huge file support.
- Segmented files permit a logical file to be transparently spread across two or more physical files.
  - Each physical file may itself be any size supported by the operating system. If the underlying operating system does not support files greater than 2 GB or 4 GB, segmented files are required for logical files to exceed these limits.
  - Segments can be located on different volumes, allowing data to expand beyond the physical limits of the hardware.
  - Most importantly, the segment definitions can be changed "on-the-fly", allowing files to grow as necessary. See Chapter 3 for more details on Segmented file support.

**Extended File Creation Block** – The key to the huge and segmented options, as well as many of the new features in the next section, is an extended header allowing the following file attributes:

- Separate specification of the first and subsequent file extension size.
- Large file size extensions, up to 2 GB.
- Definable limit on the total file size.
- Disk space availability checks on a volume or per-file basis.
- Encryption on a per-file basis.

These new attributes are defined at file creation time with the extended creation block structure, which is described in more detail in Chapter 4 of this guide.

1.2 Unicode Support and Extended Key Segments

Chapter 5 describes Unicode as implemented in **c-tree Plus**. While you could always store Unicode data in a c-tree Plus file, just as you can store any binary ‘blob’, **c-tree Plus** now supports Unicode data types, key segments, and file names.

In the course of adding Unicode to **c-tree Plus**, extended key segments were added. This capability, which will be exploited in more detail in the future, allows extended definition of key segment types on a segment, file, or application basis.

1.3 Advanced File Encryption

Advanced File Encryption expands developer options by providing a suite of protocols to protect user data:
1.4 Portability Enhancements

New platforms (Windows CE and QNX RTP) and platform-specific enhancements for Mac, Windows, Novell, HP-UX, SCO OpenServer, NetBSD, FreeBSD, and Solaris (Intel) are detailed in Chapter 6.

1.5 Enhanced Variable-Length Space Management

A new key segment mode allows the creation of an index that enhances variable-length file deleted space management, allowing c-tree Plus to merge adjacent deleted spaces and providing backward traversing of the physical file. See Chapter 8 for additional details.

1.6 Server and c-tree Plus Enhancements

The following enhancements are detailed fully in Chapter 9 and Chapter 10 of this guide:

- Transaction dependent file creates and deletes. This feature allows file creation and deletion under transaction control, including full rollback and roll-forward capability. This can be critical for systems requiring a specific set of files to be created or destroyed as a group.
- Improvements in key segment handling allow any field type to be supported in the variable-length portion of the file.
- Improved performance on multi-CPU systems due to the alignment of high-use shared objects on cache memory boundaries.
- Enhanced monitor windows for the c-tree Server.
- Improved detection of dropped or inactive client connections by the c-tree Server.
- The space allocated to files deleted from superfiles can now be reclaimed and a standalone utility can change the sector size of existing superfiles.
- Adaptive node splits – To help improve the “density” of the b+ tree structures built when key values are added in sorted order, the recent key insertion history of each buffer (which holds the contents of an index node) is automatically maintained and used to determine how to split a full node.
- File-specific Encryption – Allows a specific encryption key for each file.
- File-specific Disk-full Checks – Determine disk status on a per-file basis.
- Other Diagnostic, API, Utility, and Miscellaneous Enhancements.
1.7 c-tree Driver Enhancements

Features and enhancements for the c-tree ODBC and Crystal Reports Drivers, including DSN-less ODBC, Registry controls, and performance enhancements. See chapter 11 for additional details.

1.8 New Function Reference

The new API functions added to the c-tree Plus API fall into the following categories:

- Functions involving file creation and deletion.
  - `CreateDataFileXtd8`
  - `CreateFileXtd8`
  - `CreateIndexFileXtd8`
  - `DropIndex`
  - `PermIIndex8`
  - `PutIFileXtd8`
  - `RebuildIFileXtd8`
  - `TempIIndexXtd8`
  - `TestHugeFile`

- Functions manipulating segment definitions.
  - `SetFileSegments`

- Extended Key Segment and Unicode functions
  - `ctMBprefix`
  - `ctu16Tou8`
  - `ctu8Tou16`
  - `GetXtdKeySegmentDef`
  - `PutXtdKeySegmentDef`
  - `TransformXtdSegment`

- Enhanced Thread Message Queue Functions
  - `ctThrdQueueReadDirect`
  - `ctThrdQueueWriteDirect`

- Functions for the Server Broadcast feature.
  - `GetServerInfoXtd`

- Functions manipulating the higher order 4 bytes of an 8 byte file address.
  - `ctGETHGH`
  - `ctSETHGH`

See Chapter 12 for complete function descriptions.
1.9 Compatibility Notes

There are few significant compatibility issues between V6.x and V7.11. While V7.11 is backward compatible to V6.x, some care and preparation can smooth the transition.

Upgrading V6 Applications

As always, c-tree Plus is designed for immediate use right out of the box.

Applications written for c-tree Plus V6 that include ctreqp.h as required will compile and link properly right out of the box. Your application will function exactly as before using the new c-tree Plus V7 library. ctreqp.h automatically includes the ctv6v7.h header (described in the "#defines" section below).

Developers wanting to use huge/segmented files or other functionality made available with the Extended file format (see "Standard and Extended File Formats" below) must also:

- Use the 8 byte file creation functions to create new Extended files.
- Change IFIL definitions for indices allowing duplicates to store 8 byte offsets for indices referencing huge files.
- Convert existing data files using the ctv67 utility.

These actions are explained below:

Standard and Extended File Formats

C-tree Plus V7 supports two different, but related, file formats:

- Standard file format – The default format for all versions of c-tree Plus.
- Extended file format – A new format containing an extended header, which supports additional features described in the Extended Feature Support section below.

The new V7 extended 8 byte file creation routines (e.g., CreateDataFileXtd8, CreatelFileXtd8, etc.) create Extended files by default. Extended files are not compatible with c-tree Plus V6. However, the extended 8 byte file creation functions will create Standard files when using the ctNO_XHDRS extended file mode. The existing file creation functions (e.g., CreateDataFile, CreatelFileXtd, etc.) create only Standard files.

The c-tree Drivers read both Standard and Extended files and create Standard files.
Extended Feature Support

The following features are specified in the Extended File Creation Block. Create files with the 8 byte creation functions, (e.g., CreateDataFileXtd8, CreateFileXtd8, etc.), or convert them with ctcv67, to allow these features:

- Huge files (up to 18,000,000 terabytes)
- Segmented files (one logical file over multiple physical files)
- Transaction-dependent File Creates and Deletes
- Extended File Extent size
- File-specific Disk Full Checks
- File-specific Encryption
- Maximum File Size Limits

Operating System File Size Limits

Different platforms support different maximum file sizes. The limit is imposed by the data type for the value used to seek to a given offset in a file. Older operating systems use a 4 byte signed offset, allowing physical files up to 2 GB. Newer platforms use 4 byte unsigned offsets (4 GB) or 8 byte unsigned offsets (~18,000,000 terabytes).

On systems supporting only 4 byte signed offsets:

- Standard c-tree Plus files can grow up to 2 GB in size.
- Extended files can be segmented into multiple physical files up to 2 GB each, allowing a single logical file to grow:
  - up to 4 GB using only segmented file support (using segments <= 2 GB each)
  - up to 18,000,000 terabytes using segmented file support and huge file support (using segments <= 2 GB each)

The following platforms are presently limited to 2 GB files:

<table>
<thead>
<tr>
<th>Platform</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AT&amp;T SVR4</td>
<td>SunOS</td>
</tr>
<tr>
<td>IBM AIX 3.2-4.1</td>
<td>Linux (before kernel 2.400)</td>
</tr>
<tr>
<td>Macintosh (7-9, X)</td>
<td>QNX</td>
</tr>
<tr>
<td>OS/2</td>
<td>SGI-Irix</td>
</tr>
<tr>
<td></td>
<td>LynxOS</td>
</tr>
<tr>
<td></td>
<td>Novell NetWare 4 and below</td>
</tr>
<tr>
<td></td>
<td>SCO OpenServer/UnixWare</td>
</tr>
<tr>
<td></td>
<td>HP-UX 10</td>
</tr>
</tbody>
</table>

On systems supporting a 4 byte unsigned offset:

- Standard c-tree Plus files can grow up to 4 GB in size.
- Extended files using huge file support and segmented file support can grow to 18,000,000 terabytes (using segments <= 4 GB each).

The following platforms fall into this category:

<table>
<thead>
<tr>
<th>Platform</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 95 and above – FAT file system</td>
<td>Novell NetWare 5</td>
</tr>
<tr>
<td>Solaris 2.6 (Intel/SPARC)</td>
<td></td>
</tr>
</tbody>
</table>
On systems supporting 8 byte offsets:
- Extended format files with huge file support can grow up to 18,000,000 terabytes.
- Segmented file support is optional, but convenient for allocating portions of files to different volumes.
- Standard c-tree Plus files can grow up to 4 GB in size.

The following platforms fall into this category:

<table>
<thead>
<tr>
<th>Windows NT/2000 – NTFS file system</th>
<th>HP-UX 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX 4.2 and above</td>
<td>Solaris 7 and above (Intel/SPARC)</td>
</tr>
<tr>
<td>Tru64 Unix</td>
<td>FreeBSD</td>
</tr>
<tr>
<td>NetBSD</td>
<td>Linux (kernel 2.4.00 or later)</td>
</tr>
</tbody>
</table>

**NOTE:** The option to support huge files (8 byte offsets) is optional, not default, on some operating systems. Enable huge file support for each volume before creating huge files on that volume under AIX, Solaris, and HP-UX. Use NTFS volumes under Windows NT and 2000.

**Duplicate Keys**

For indices associated with Standard data files, a duplicate key length includes 4 bytes for the associated record position which is used to break ties. If an index is created for a HUGE data file, then the key length must include 8 bytes for the associated record position.

**CTCV67 - Extended File Conversion Utility**

`ctcv67 <old file name> <new path> [option <argument> ...]`

`ctcv67` performs various conversions, including:
- Converting data and index files from Standard c-tree Plus format to Extended c-tree Plus format (huge or non-huge) using Incremental ISAM (IFIL) or parameter files.
- Adding huge file support to a non-huge file.
- Adding, removing, or changing segment support.
- Automatically compacting the output file.

If the original file has an extended header, the new file inherits the extended header attributes. This utility does not convert a huge file to a non-huge file. If a superfile host is specified, `ctcv67` converts all member data files and their associated indices. When a mirrored file is specified, the conversion does NOT produce the mirrors, but maintains the mirrored names in IFIL structures. Copy the converted master files to produce the mirror files.

Unless a parameter file is used, a data file must have an embedded IFIL definition for its associated indices to be converted. A data file will have an embedded IFIL
definition if it was created with one of the `CreatelFile` routines or `PutlFile` was called for the file.

If an index file is specified by the `<old file name>` parameter, then a new index file with an extended header will be created, all the resources will be transferred to the new index, its key length will be increased if HUGE is specified and the key supports duplicates, but no key values will be transferred to the new file. In essence, it will be an empty index file.

The following options are available:

<table>
<thead>
<tr>
<th>Option</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>&lt;yes</td>
<td>no&gt;</td>
</tr>
<tr>
<td>P</td>
<td>&lt;page size&gt;</td>
<td>Sets the page size in bytes used to create the new files. This setting must be compatible with the existing file. Default: 2048</td>
</tr>
<tr>
<td>H</td>
<td>&lt;yes</td>
<td>no&gt;</td>
</tr>
<tr>
<td>T</td>
<td>&lt;yes</td>
<td>no&gt;</td>
</tr>
<tr>
<td>G</td>
<td>&lt;new segment host file name&gt; &lt;host segment size in MB&gt;</td>
<td>The new file name of a data or index file to be created as a segmented file. Derive this name from the new path and the file’s original name. (The &quot;T&quot; option can be used to check on the new name assigned to a file.) Use a G entry for each new file to be segmented. <strong>NOTE:</strong> Even if the old file is segmented, segments must be specified for the new file or it will no longer be segmented.</td>
</tr>
<tr>
<td>S</td>
<td>&lt;segment name&gt; &lt;segment size in MB&gt;</td>
<td>This option must follow immediately after the G option for the associated host file. Repeat this option as necessary to create all the additional segments of the file. The segment name will be used exactly as specified. The new path will NOT be prepended to the name.</td>
</tr>
</tbody>
</table>

Ordinarily, the new path name is simply prepended to the original file name. If both have absolute names, the new (absolute) path replaces the absolute portion of the original name. If only the original file is absolute, the new (relative) path is inserted.
immediately after the absolute portion of the original file name. Otherwise, the new path (relative or absolute) is prepended to the original (relative) file name.

For example, to convert the data file \C:\DATA\SAMPLE.DAT and its associated index, \C:\DATA\SAMPLE.IDX, into huge files with three segments of 2000 MB each, use the following command line:

```
ctcv67 C:\DATA\SAMPLE.DAT C:\HUGEDATA\ H yes
G C:\HUGEDATA\DATA\SAMPLE.DAT 2000 S D:\HUGEDATA\SAMPLE.SG1
2000 S E:\HUGEDATA\SAMPLE.SG2 2000 G
C:\HUGEDATA\DATA\SAMPLE.IDX 2000 S D:\HUGEDATA\SAMPLE.IS1
2000 S E:\HUGEDATA\SAMPLE.IS2 2000
```

Even if the original file is segmented, the new file will not be segmented unless a new segment definition is specified using the "G" and "S" options.

The associated indices are recreated based on the IFIL resource embedded in the data file or based on the parameter file information. If the associated data file becomes huge, index files allowing duplicate records will have their key lengths automatically increased by 4 bytes to accommodate the longer associated position information used to break the tie. If you recall, when allowing duplicate key values in an index, you must add 4 bytes to the key length. For huge files, you must add a total of 8 bytes. In addition, the index file names in the IFIL definition will reflect the new path name.

`ctcv67` creates (or appends to) a text file named `CV67.REP`, which reports on the actions and progress of the utility program.

---

**#defines**

The **c-tree Plus** #define constants were modified to use a uniform style of `ctWXYZ` by appending `ct` to the beginning of all **c-tree Plus** defines. For example, `TRNLOG` is changed to `ctTRNLOG`.

Two new header files, `ctv6v7.h` and `ctv7v6.h`, provide compatibility.

Add the `ctv6v7.h` compatibility header file to applications written for **c-tree Plus** V6.x to automatically change the old defines to the new defines to allow the application to be compiled with **c-tree Plus** V7.11.

Add the `ctv7v6.h` compatibility header file to applications written for **c-tree Plus** V7.11 to automatically change the new defines to the old defines to allow the application to be compiled with **c-tree Plus** V6.x. This header provides "rename #defines" to set the symbols back to their V6 convention.
The following defines were changed:

<table>
<thead>
<tr>
<th>ADMOPEN</th>
<th>LKSTATE</th>
<th>RESTRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG_LOCK</td>
<td>LOGFIL</td>
<td>RESTRED_BLK</td>
</tr>
<tr>
<td>AUTOSAVE</td>
<td>LOGIDX</td>
<td>RESTRSV</td>
</tr>
<tr>
<td>AUTOTRN</td>
<td>MIRROR_SKP</td>
<td>SAVECTREE</td>
</tr>
<tr>
<td>CHECKLOCK</td>
<td>NONEXCLUSIVE</td>
<td>SAENV</td>
</tr>
<tr>
<td>CHECKREAD</td>
<td>OPENCRIPT</td>
<td>SHARED</td>
</tr>
<tr>
<td>CIPFASE</td>
<td>OVRFASE</td>
<td>SS_LOCK</td>
</tr>
<tr>
<td>COMMIT_SWAP</td>
<td>PENDERR</td>
<td>SUPERFILE</td>
</tr>
<tr>
<td>DEFERCP</td>
<td>PENDREN</td>
<td>SUSPEND</td>
</tr>
<tr>
<td>DELUPDT</td>
<td>PENDRNU</td>
<td>TRANDEP_CRE</td>
</tr>
<tr>
<td>DISABLERDT</td>
<td>PERMANENT</td>
<td>TRANDEP_DEL</td>
</tr>
<tr>
<td>DUPCHANEL</td>
<td>PREIMG</td>
<td>TRANDEP_SFM</td>
</tr>
<tr>
<td>ENABLE</td>
<td>READFIL</td>
<td>TRNBEGLK</td>
</tr>
<tr>
<td>ENABLE_BLK</td>
<td>READREC</td>
<td>TRNLOG</td>
</tr>
<tr>
<td>EXCLUSIVE</td>
<td>READREC_BLK</td>
<td>TWOFASE</td>
</tr>
<tr>
<td>FREE</td>
<td>RESET</td>
<td>VIRTUAL</td>
</tr>
<tr>
<td>GETLKISAM</td>
<td>RESTORE</td>
<td>VLENGTH</td>
</tr>
<tr>
<td>LK_BLOCK</td>
<td>RESTORE_BLK</td>
<td>WRITETHRU</td>
</tr>
</tbody>
</table>

Existing V7.11 Limitations

- **RenameFile** and **RenameIFile** are not supported for segmented files.
- There is no way to pass segment definitions to **RebuildIFileXtd8**, **PermIIndex8**, or **TemplIndexXtd8**, but **RebuildIFileXtd8** can use the existing segment definitions in the host segment if they are not corrupted.
- Huge and Segmented file support is limited in heterogenous Standalone Multi-user environments that include Win32 platforms. This is due to the different locking standards between platforms. The standard for huge and segmented file multi-user operation is the client/server model.
- FairCom’s software level mirroring logic cannot be used on segmented files.
- **IMPORTANT**: **SetFileSegments** must not be called from **within an active transaction** except for the first call after a transaction dependent create.
- **CompactIFileXtd8**, which will compact Extended files, is not implemented. To compact Extended files in the interim, use the **ctcv67** standalone utility.
2. Huge File Support

When c-tree Plus was created, 10MB hard drives were considered large. A 2 GB maximum file size was sufficient, if not unimaginable.

More than 20 years have passed. With 64 bit operating systems becoming common and 4GB hard drives being considered bottom-of-the-line, the need has developed to support 8 byte addresses for file offsets. This provides a maximum file size of \((4GB)^2\), or more than 1.844 X 10^{19} bytes, per file.

FairCom supports huge files in two ways:

- **huge files** - 8 byte offsets as described in this chapter, and
- **segmented files** - logical files distributed across multiple physical files, as described in the following chapter.

**NOTE:** Huge and Segmented file support is limited in heterogenous Standalone Multi-user environments that include Win32 platforms. This is due to the different locking standards between platforms. The standard for Huge file multi-user use is the client/server model.

### 2.1 Huge File Basics

The following step-by-step instructions walk you through the process of using huge files in general. Details are added throughout this chapter and an example is included in section 2.3.

1. Create a library supporting huge files. The default `ctHUGEFILE` define activates both Huge File and Segmented File support in the c-tree Plus client libraries.
2. Use the new Xtd8 creation functions described in the "Xtd8 Creation Functions" section of this chapter and further defined in the function descriptions:
   a. Use the `ctFILEPOS8` extended file mode in the `XCREblk` structure, described later in this chapter.
   b. Create an array of `XCREblk` structures, one for each physical file to be created.
   c. Create the file(s) using an Xtd8 create function using the `XCREblk` array.

**NOTE:** Any index referencing a data file created using 8 byte file addresses must also use 8 byte file addresses. A `ctFILEPOS8` data file requires a `ctFILEPOS8` index. A `ctFILEPOS8` index supporting duplicate keys must allow for 8 bytes in its key length for the automatic tiebreaker that c-tree Plus automatically appends to the key value. See the example for more details.

The Xtd8 create functions always create Extended files, even if no new features are requested (unless the `ctNO_XHDRS` file mode is turned on). Files created with the original API (e.g., `CreateFileXtd`) are in the Standard c-tree Plus format. A c-tree Plus V6 application receives a `FVER_ERR(43)` when attempting to open an Extended file.
NOTE: Files are created in ctEXCLUSIVE mode, so you must close and reopen the file after it is created to allow it to be shared. Since files are created in ctEXCLUSIVE mode, this is a convenient time to execute additional configuration functions, such as SetFileSegments and PutDODA.

2.2 Xtd8 File Creation Functions

The extended 8 byte file creation functions are the same as their 4 byte counterparts except that a new parameter has been added: a pointer to the XCREblk structure, or an array of XCREblk structures, defined in the "Extended File Creation Block Structure" chapter. For example, the 4 byte create data file extended function:

```
COUNT CreateDataFileXtd(COUNT datno, pTEXT filnam,
UCOUNT datlen, UCOUNT xtdsiz, COUNT filmod,
LONG permmask, pTEXT groupid, pTEXT fileword)
```

becomes the extended 8 byte create data file function:

```
COUNT CreateDataFileXtd8(COUNT datno, pTEXT filnam,
UCOUNT datlen, UCOUNT xtdsiz, COUNT filmod,
LONG permmask, pTEXT groupid, pTEXT fileword,
pXCREblk pxcreblk)
```

`pxcreblk` points to a single XCREblk structure, defined later in this chapter.

A create function referencing more than one file, such as CreateIFileXtd, is modified in the same manner, except the `pxcreblk` parameter points to an array of XCREblk structures, one for each physical file referenced by the function. For example,

```
COUNT CreateIFileXtd(pIFIL ifilptr, pTEXT dataextn,
pTEXT indxextn, LONG permmask, pTEXT groupid,
pTEXT fileword)
```

becomes

```
COUNT CreateIFileXtd8(pIFIL ifilptr, pTEXT dataextn,
pTEXT indxextn, LONG permmask, pTEXT groupid,
pTEXT fileword, pXCREblk pxcreblk)
```

If the IFIL structure pointed to by `ifilptr` describes a data file with 10 indices, but the indices are contained in two physical index files, `pxcreblk` must point to an array of three XCREblk structures, one for each physical file. If any of these files do not require the extended features, the corresponding XCREblk structure is zero filled.

The complete list of new file creation functions includes: CreateDataFileXtd8, CreateIFileXtd8, CreateIndexFileXtd8, PermIIndex8, RebuildIFileXtd8, and TempIIndexXtd8. These functions are detailed in the Function Description chapter in these notes.
2.3 Huge File Creation Example

This example demonstrates creation of ISAM files supporting 8 byte file addresses. For simplicity, it uses a single data file with a single index with a single segment.

/* define extended file attributes */
XCREblk creblks[2] = {
    { ctFILEPOS8, /* x8mode: support HUGE data file */
      0, 0, 0, 0,
      1048576}, /* lxtsiz: 1MB file extensions */
    { ctFILEPOS8, /* x8mode: support HUGE index file */
      0, 0, 0, 0,
      1048576} /* lxtsiz: 1MB file extensions */
};

/* define ISAM attributes */
ISEG seg = { 4, 6, 0};
IIDX idx = { 14, 4, 1, 0, 0, 1, &seg};

/* Note that the segment is 6 bytes, but the key allows 
 * duplicates, so the key length is 14, allowing 8 bytes 
 * for the offset tiebreaker. */

IFIL fil = { "bigdata", -1, 128, 0,
            ctSHARED | ctVLENGTH | ctTRNLOG, 1, 0,
            ctSHARED | ctVIRTUAL | ctTRNLOG, &idx};

InitISAM(10,5,32);  /* Initialize c-tree Plus */

/* create HUGE ISAM files */
CreateIFileXtd8( &fil, NULL, NULL, 0L, NULL, NULL,
                  creblks /* pointer to array of XCREblks */
);
2.4 Record Offsets Under Huge File Support

c-tree Plus does not rely on native 64 bit (8 byte) integer support. Creating files greater than 4GB without segmented files requires a file I/O library supporting 64 bit file addresses; but this is a different issue from native 64 bit integer support, which means the operating system uses 64 bit addresses.

c-tree Plus uses the standard c-tree Plus 32 bit oriented API with both 32 bit files and 64 bit files. However, with 64 bit files, two calls handle the higher order 32 bits (4 bytes): ctSETHGH and ctGETHGH. These routines are only needed with ISAM and low-level functions that return or use as input explicit file positions. For example, AddKey has an input parameter that passes an explicit record address, and FirstKey returns a record address. These functions need the additional API calls. AddRecord and FirstRecord do not use explicit record addresses and do not need these additional API calls, but CurrentFileOffset does.

The ctSETHGH routine, NINT ctSETHGH(LONG highword), is called before a low-level routine requiring a record address as an input parameter. ctSETHGH always returns NO_ERROR. ctGETHGH, LONG ctGETHGH(VOID), is called after a low-level routine that returns or sets an output parameter to a record address.

To minimize the effect on performance in client-server environments, ctGETHGH and ctSETHGH do not make separate calls to the c-tree Server. Instead, the information needed by ctGETHGH or supplied by ctSETHGH is cached on the client side.

For sample code, see the example in the ctSETHGH function description. See the ctSETHGH and ctGETHGH function descriptions for more information.
3. Segmented File Support

Segmented file support allows a single logical file to occupy multiple physical files. This allows data files to exceed the physical file size limits imposed by the operating system, and when combined with Huge File Support, provides the added benefit of very large file sizes. The physical file segments can be kept together or distributed over multiple volumes.

**NOTE:** Huge and Segmented file support is limited in heterogenous Standalone Multi-user environments that include Win32 platforms. This is due to the different locking standards between platforms. The standard for Huge file multi-user use is the client/server model.

3.1 Segmented File Basics

Files can be segmented automatically or you can choose the size and location of specific segments. The following step-by-step instructions will walk you through the process of using segmented files in general. Details are added throughout this chapter.

1. Create a library supporting huge files. The `ctHUGEFILE` define activates both Huge File and Segmented File support in the `c-tree Plus` client libraries by default.

2. Use the new Xtd8 file creation functions described in section 2.3 and further defined in the function descriptions.
   a) The `ctFILEPOS8` file mode permits huge files. This mode is required if the logical file will exceed 4GB total file size, but is not required for segmented files in general.
   b) The `ctSEGAUTO` file mode allows automatic segment generation. See “Automatic Segments” below for more details.
   c) Generate an `XCREblk` structure for each physical file to be created.

3. Use `SetFileSegments` to establish the initial segments. This step is NOT required with automatic segment generation.
   a) Define a `SEGMDEF` structure detailing the segments to be used.
   b) Execute `SetFileSegments` on the file with the `SEGMDEF` structure.

The extended creation functions can only provide minimal information for segmented files. The `XCREblk` structure only holds the size of the host segment and the maximum number of segments. To fill in the details, the `SetFileSegments` function, short name `ctSETSEG`, specifies segment definitions for newly created files dynamically while the file is open. Also, `SetFileSegments` can optionally set or change the size limit on the host segment. However, `SetFileSegments` can only be called for files created with the new Xtd8 API, which causes the file to have an extended header.

The Xtd8 create functions always create Extended files, even if no new features are requested (unless the `ctNO_XHDRS` file mode is turned on). Files created with the
original API (e.g., `CreateFileXtd`) are in the Standard c-tree Plus format. A c-tree Plus V6 application receives a `FVER_ERR(43)` when attempting to open an Extended file.

**NOTE:** Files are created in `ctEXCLUSIVE` mode, so you must close and reopen the file after it is created to allow it to be shared. Since files are created in `ctEXCLUSIVE` mode, this is a convenient time to execute additional configuration functions, such as `SetFileSegments` and `PutDODA`.

### 3.2 Automatic Segments

The `ctSEGAUTO` file mode allows automatic segment generation. When this file mode is used, `SetFileSegments` is not required. Simply set the host segment size and maximum segments. All additional segments will be the same size as the host file and will be stored in the same directory. The segment names generated for a file start by adding ".001" to the existing file name, then incrementing the numeric extension. For example: The automatic segment names for the file `sample.dat` would start with `sample.dat.001` and continues with `sample.dat.002`, and so on.

This option is supported by the `ctcv67` conversion utility.

### 3.3 SEGMDEF Structure

The segmented file specifications in a call to `SetFileSegments` are expressed through the `SEGMDEF` structure, which is made up of a pointer to a segment name and a 4 byte integer holding the **segment size in MB**. The segment definitions are stored in the host, or first, file segment as a special FairCom resource.

There is no limit on the number of segments except that the ASCII strings containing the segment names and sizes are limited to just over 8100 bytes. This means that even if each segment requires a 255-byte file name, a minimum of 30 segments could still be defined within the size constraint of the special resource.

The segment definitions can be changed on the fly, even while the file is being updated.

A `SEGMDEF` structure is used to specify each segment. It is defined in `ctport.h` as:

```c
typedef struct segmdef {
    pTEXT sgname; /* pointer to segment name */
    LONG sgsIZE; /* size of segment in MB */
} SEGMDEF;
```

`sgname` points to the name of the segment. The `sgname` parameter conveys not only a name to use for the segment, but, by virtue of any path or device specification used in the segment name, where the segment should be located.

`sgsize` specifies the size of a segment in MB. Therefore, if you have the segment size in bytes, divide it by 1,048,576 to get MB.
3.4 SetFileSegments Function

The **SetFileSegments** function is defined as follows:

```
COUNT SetFileSegments(COUNT filno, NINT aseg, NINT tseg, pSEGMDEF pseg)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filno</code></td>
<td>The file number of an open file</td>
</tr>
<tr>
<td><code>aseg</code></td>
<td>Specifies the number of active segments, i.e., the segments created immediately. <code>aseg</code> must be at least one and less than or equal to <code>tseg</code>.</td>
</tr>
<tr>
<td><code>tseg</code></td>
<td>Specifies the total number of segments pointed to by <code>pseg</code>.</td>
</tr>
<tr>
<td><code>pseg</code></td>
<td>Points to an array of <code>SEGMDEF</code> structures described below.</td>
</tr>
</tbody>
</table>

If the first segment definition pointed to by `pseg` has an `sgname` pointing to the empty string, i.e., `*sgname == '\0'`, not `sgname == NULL`, the `sgsize` member of the structure becomes the host segment size limit. Only the last segment can have a size limit of zero, which is interpreted as no limit. If the first segment definition pointed to by `pseg` does not have an `sgname` pointing to the empty string, then this `sgname` applies to the second segment.

Additional segments automatically become active as needed, up to the maximum set in `tseg`. The segments are used in the order defined by the array of `SEGMDEF` structures pointed to by `pseg`.

The file referenced by `filno` must be opened in `ctEXCLUSIVE` mode the first time `SetFileSegments` is called. Note that a file which has been created and not yet closed is always in `ctEXCLUSIVE` mode, regardless of the file mode specified as part of the create call. After the segment definitions have been established by the first call to `SetFileSegments`, it is possible to call `SetFileSegments` to modify the segment definitions even while the file is being updated in client/server models. However, it is not possible to change a segment size so that the new size is smaller than the actual physical size of the segment, nor can a segment size be increased if the segment has already reached its previously specified size, nor can `SetFileSegments` rename a segment that is in use. A segment is in use if data beyond the segment header information has been written to the segment. An active segment is not in use just because it is on disk; data must have been written to it. Therefore, a call to `SetFileSegments` can, in real time, change where segments will reside (provided the segment is not already in use) and/or how large they are (provided the new size is not smaller than the current physical size or that the segment has already been completely filled with data).

**NOTE:** The `fxtsiz` member of the `XCREblk` structure cannot be set higher than the size of the first (host) segment during a file create. This will result in a `SEGM_ERR(674)` error signifying the need for more segments, which do not exist yet because `SetFileSegments` has not yet been called.
3.5 File Segment Example

In this example, the segmented file consists of a host segment limited to 500 MB, a second segment limited to 1 GB, and a final segment unlimited in size. However, the file will not be a huge file, so the total size will be limited to 4 GB.

XCREblk creblk = {
  0, /* FILEPOS8 is NOT on: not HUGE */
  500, /* 1st segment size is 500MB */
  0, /* no specified file size limit (HW) */
  0, /* no specified file size limit (LW) */
  104857600, /* file created at size 100MB */
  10485760, /* file extend 10MB at a time */
  3, /* maximum number of segments is 3 */
  0 /* no disk full threshold */
};

SEGMDEF segdef[2] = {
  "d:dataseg.2",1024, /* 1024MB = 1GB size limit */
  "e:dataseg.3",0, /* no limit on segment size */
};

/* create data file, specifying the host segment size */
CreateDataFileXtd8 (
  10, /* data file number */
  "c:hostseg.dat", /* data file name */
  384, /* record length */
  0, /* creblk specifies large extent sizes */
  TRNLOG, /* support transaction processing */
  0L, /* no permission mask */
  NULL, /* no group ID */
  NULL, /* no password */
  &creblk); /* pointer to extended create block */

/* specify definitions for the two other segments */
SetFileSegments(
  10, /* data file number */
  1, /* one active segment (the host segment) */
  2, /* two segment definitions to be passed */
  segdef); /* pointer to the segment definitions */
3.6 Segmented Dynamic Dump

The c-tree Server V7 and the ctdump and ctrdmp utilities support dynamic dumping of segmented files and the creation of segmented (stream) dump files. This is different from the feature introduced in V6.x that automatically broke the dump file into 1GB 'extents'. Dumping to segmented files allows you to take advantage of huge file support and to specify the files size and location for each dump file segment.

- To dump segmented files, simply list the host (main) file in the !FILES list and the segments will be managed automatically.

- To cause the output dump file produced by the dynamic dump itself to be segmented, use these new script entries:

```plaintext
!SEGMENT <size of host dump file in MB>
<dump seg name> <size of segment in MB>
...
<dump seg name> <size of segment in MB>
!ENDSEGMENT
```

The host dump file is the file specified in the usual !DUMP entry. Only the last segment in the !SEGMENT / !ENDSEGMENT list can have a zero size specified, which means unlimited size.

3.7 Segmented Dynamic Dump Example

Assume `bigdata.dat` is a segmented file with segment names `bigdata.sg1`, `bigdata.sg2`, and `bigdata.sg3`, and the index file `bigdata.idx` is not segmented. To dump these files into a segmented dump file, use the script:

```plaintext
!DUMP bigdump.hst
!SEGMENT 50
    bigdump.sg1 75
    bigdump.sg2 0
!ENDSEGMENT
!FILES
    bigdata.dat
    bigdata.idx
!END
```

The host dump file is up to 50 MB, the first dump file segment is up to 75 MB, and the last dump file segment is as large as necessary.
4. Extended File Creation Block Structure

The key to many of the new features introduced in **c-tree Plus V7** is the extended header in Extended files. With the exception of the specification of segmented files, all of the new file definition capabilities are specified through a new parameter block, the **XCREblk** structure, containing the additional information needed to implement the new header.

A file supporting 8 byte addresses and/or which uses any of the new file attributes uses an extended file header instead of the 128 bytes used by Standard 4 byte **c-tree Plus** files. Some of the additional bytes are used for the contents of the **XCREblk** structure, made up of sixteen 4 byte integers, seven of which are reserved for future use. The **XCREblk** structure passes the new file attributes to the file creation routines.

```c
typedef struct XCREblk {
    LONG x8mode;  /* new file modes */
    ULONG segsiz; /* host segment size (MB) */
    LONG mxfilzhw; /* high word max file size */
    LONG mxfilzlw; /* low word max file size */
    LONG fxtsiz;  /* first file extent size */
    LONG lxtsiz;  /* file extent size */
    LONG segmax;  /* maximum number of segments */
    ULONG dskful; /* disk full threshold */
    ULONG filkey; /* file encryption key */
    LONG rs6[6];  /* reserved for future use */
    LONG callparm; /* call specific parameter */
} XCREblk;
```

### 4.1 Extended File Modes (x8mode)

During file creation, the **x8mode** member of the **XCREblk** structure can specify new file modes defined in **ctport.h**:

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctFILEPOS8</td>
<td>0x000040</td>
<td>Huge file support.</td>
</tr>
<tr>
<td>ctFILESEG</td>
<td>0x00008</td>
<td>Segmented file. It is not necessary to specify ctFILESEG. Setting the host segment's size limit, <strong>segsiz</strong>, or calling <strong>SetFileSegments</strong>, automatically sets this mode.</td>
</tr>
<tr>
<td>ctNO_XHDRS</td>
<td>0x10000</td>
<td>Forces Standard header instead of Extended.</td>
</tr>
<tr>
<td>ctNOENCRYP</td>
<td>0x00100</td>
<td>If file encryption has been enabled by a call to <strong>SetEncryption</strong>, an individual file can disable encryption by or'ing ctNOENCRYP into the <strong>x8mode</strong> member of the <strong>XCREblk</strong> structure.</td>
</tr>
</tbody>
</table>
Symbolic Constant | Value | Explanation
--- | --- | ---
ctNOSPACHK | 0x00080 | Turn off disk full checking for this file only.
ctSEGAUTO | 0x200000L | Use up to segmax automatic file segments of segsiz MB. SetFileSegments is not required.
ctTRANDEP | 0x00010 | Transaction Dependent Create and Delete.
ctRSTRDEL | 0x00800 | Restorable Deletes. All the features of ctTRANDEP plus the ability to roll back file deletions, recovering older versions of the file.

**Host Segment Size (segsiz)**

The segsiz member of the XCREblk structure permits the host's segment size to be specified.

**NOTE:** Segment sizes are specified as multiples of a megabyte: 1048576 bytes. For example, to specify that the host segment size will be one gigabyte, 1 GB, set segsiz to 1024. It is not necessary to specify the host segment size in XCREblk since SetFileSegments can optionally set the host segment size as well as the other segment sizes.

**Maximum Number of Segments (segmax)**

The segmax member of the XCREblk structure specifies the maximum number of segments permitted for the file, including the host segment. This value can be overridden with SetFileSegments.

**Max File Size Limit (mxfilzhw, mxfilzlw)**

The mxfilzhw and mxfilzlw members of the XCREblk structure permit a maximum file size limit to be specified for a 4 byte or 8 byte file. The file size is specified in bytes. For files over 4 GB, mxfilzhw must be a non-zero value. For each multiple of 4 GB, add one to the mxfilzhw member. For example, to limit the file size to 8 GB, set the (mxfilzhw, mxfilzlw) pair to (2,0). To limit the size to 8.5GB, use the pair of values (2, 2147483648). To indicate no file size limit, set both values to zero.
Extended File Creation Block Structure
Extended File Modes (x8mode)

File Extent Size (lxtsiz, fxtsiz)

c-tree Plus V6.x files are limited to a file extension size of under 64KB. The lxtsiz member of the XCREblk structure can be used to define file extension sizes of up to 2 GB. lxtsiz is specified in bytes. The fxtsiz member specifies the first file extension size applied when the file is created and may be up to 2 GB. For example, to start a file with an initial size of 100 MB, and to have the file grow by an additional 10 MB as more space is required, use these values:

<table>
<thead>
<tr>
<th>fxtsiz</th>
<th>100 * 1024 * 1024 = 104857600</th>
</tr>
</thead>
<tbody>
<tr>
<td>lxtsiz</td>
<td>10 * 1024 * 1024 = 10485760</td>
</tr>
</tbody>
</table>

Individual File Disk Space Threshold (dskful)

All of the Disk Full features of c-tree Plus V6.x apply to V7, plus the ability to turn on/off disk full checking on a file-by-file basis and the ability to define the disk full threshold on a file-by-file basis.

To turn off disk full checks for a particular file, create the file using the 8 byte extended creates (e.g., CreateDataFileXtd8) and set the ctNOSPACHK bit in the x8mode member of the XCREblk structure on.

To turn on file by file checking with a file specific disk full threshold, set the dskful member of the XCREblk structure to the desired limit at file create time using the extended 8 byte create functions.

To specify the size, in bytes, of the disk-full threshold to be used whenever the file is extended, place the non-zero limit in the dskful member of the XCREblk structure. If extending the size of the file will leave less than the threshold specified in dskful, then the write operation causing the file extension will fail with the error SAVL_ERR(583).

File Encryption Key (filkey)

Whether or not SetEncryption has been called, setting the filkey member of the XCREblk structure to a non-zero value causes the file to be encrypted using the value of the filkey member as the encryption key.
5. Unicode Support

This chapter describes the c-tree Plus support for Unicode. This support includes:

- Storing Unicode data
- Indexing Unicode data
- Recognizing Unicode file names

This support is in addition to our ordinary string support, not an either/or approach.

The final section also describes, in general, the extended key segment support, which will allow support for other types of complex key segments in the future.

5.1 Unicode Concepts

Unicode is an effort to standardize the representations of all languages in computer format. Early standards, like ASCII, only encoded letters for English. Efforts to internationalize started with extending ASCII to include characters used in other western languages, such as umlauts and accents, but was limited by a 255 character set that would fit in one byte. Unicode incorporates the characters of all the major government standards for ideographic characters from Japan, Korea, China, and Taiwan, and more.

Though Unicode is thought of as a wide-character encoding with 16 bits per character, Unicode standards include 8 bit multi-byte encoding (UTF8), 16 bit wide character encoding (UTF16), and 32 bit wide character encoding (UTF32). c-tree Plus supports both UTF8 and UTF16.

Well-defined conversion routines permit unambiguous translation among UTF8, UTF16, and UTF32.

A Unicode string is terminated by a null character: a single zero byte for UTF8, and 2 and 4 zero bytes for UTF16 and UTF32, respectively.

NOTE: UTF16 does not encode all characters with a single 16 bit code unit. There are some languages that incorporate a sequence of two 16 bit code units to encode a single character.

5.2 Preparation

Unicode support is currently available for any client when connecting to the c-tree Server for Windows and for c-tree Plus Standalone libraries under Windows. For client operation, ensure you install the c-tree Server for Windows with Unicode support.

When building the c-tree Plus libraries, execute mtmake with the “u” flag, “mtmake u”, to prepare the library for Unicode support. Standalone builds need the ICU libraries from the ICU web site, as described in section 5.4 below.
5.3 Storing Unicode Data

Simply storing Unicode data has always been possible with c-tree Plus, provided the application treated the data as binary and performed any necessary translations. Index support is described in the next section, “Unicode Key Segments”.

Storing UTF16 Data

Storing Unicode data requires DODA entries for each field. The individual wide-characters used in UTF16 are not platform independent with respect to byte ordering. They are treated the same as short integers: on LOW_HIGH platforms, the lower order byte comes before the higher order byte. With the DODA entries in place, the Server and clients manage byte-order translation.

c-tree Plus has four Unicode (UTF16) field types:

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT_FUNICODE</td>
<td>A fixed length field containing a UTF16 encoded, null terminated string.</td>
</tr>
<tr>
<td>CT_UNICODE</td>
<td>A variable length field containing a UTF16 encoded, null terminated string.</td>
</tr>
<tr>
<td>CT_F2UNICODE</td>
<td>A fixed length field that begins with a 2-byte integer specifying the number of bytes in the following UTF16 encoded string.</td>
</tr>
<tr>
<td>CT_2UNICODE</td>
<td>A variable length field that begins with a 2-byte integer specifying the number of bytes in the following UTF16 encoded string.</td>
</tr>
</tbody>
</table>

NOTE: The length fields at the beginning of CT_F2UNICODE and CT_2UNICODE are specified in bytes. Specifying a field length in bytes is consistent with all other c-tree Plus field types, but it is inconsistent with system level routines that ordinarily use number of characters, not number of bytes, to describe the length of wide-character strings.

Storing a UTF16 string longer than 64KB requires a CT_UNICODE field. To store a string greater than 64KB with a length prefix, convert the string to UTF8 and store it in a CT_4STRING field, as discussed below.

Storing UTF8 Data

Since a UTF8 encoded string is comprised of ordinary ASCII characters (with code values between 0 and 127), and multi-byte characters (which have the highest-order bit set in each byte), they can be stored normally in a c-tree Plus record when a DODA is not present. It is simply up to the application to decipher the record, as with any other data type.

With a DODA present, store UTF8 encoded strings in any c-tree Plus standard string type, such as CT_STRING, CT_4STRING, etc. Since c-tree Plus only interprets the contents of a field when the field is part of a key value, storing a UTF8 string in an “ordinary” c-tree Plus string-type field works, provided:
• Indexing is not required, or
• There is a mechanism to permit c-tree Plus’s key assembly routine to properly interpret the string field. c-tree Plus’s new extended key segment capability deals with this situation.

C-tree Plus provides conversion routines between UTF8 and UTF16. Contact FairCom if you require routines to handle UTF32 conversion. The input strings are assumed to be terminated by a NULL character. All output buffer sizes are specified in bytes. The conversion routines return NO_ERROR (0) on success, VBSZ_ERR (153) if the output buffer is too small, or BMOD_ERR (446) if there is a problem with the input string.

• ctu8TOu16 converts an ASCII or UTF8 encoded string to a UTF16 Unicode string.

\[\text{NINT ctu8TOu16(pTEXT u8str,pWCHAR u16str,VRLEN u16byt)}\]

• ctu16TOu8 converts in the other direction.

\[\text{NINT ctu16TOu8(pWCHAR u16str,pTEXT u8str,VRLEN u8byt)}\]

5.4 Unicode Key Segments

Unicode key segments provide a challenge for two reasons:
1) Unlike all other key segments previously implemented, the number of bytes stored in the key and the number of bytes of source data used to construct the key are not the same.
2) The derivation of the binary sort key (segment) stored in the index from the source data is not a simple transformation.

To accommodate both of these challenges, c-tree Plus incorporated "extended key segments." The concept of an extended key segment can be applied to virtually any non-standard key segment. Our first implementation is for Unicode keys.

Because of the complexity of the Unicode collation algorithm, and because of the incredible breadth of language and country support envisaged by Unicode, FairCom has chosen to implement Unicode key segments using the International Components for Unicode (ICU) open-source development project. The ICU implementation of Unicode support is available on a wide variety of platforms, but not every platform. The ICU website can be accessed at:

http://oss.software.ibm.com/developerworks/opensource/icu/project/

The ICU projects use the X open-source license. It allows ICU to be incorporated into a wide variety of software projects using the GPL license. The X license is non-viral, allowing ICU to also be incorporated into non-open source products. The X license is a free software license that is compatible with the GNU GPL license (see http://www.gnu.org/philosophy/license-list.html#GPLCompatibleLicenses).
The text of the X license is available at http://www.x.org/terms.htm, but is included below:

**Terms and Conditions**

COPYRIGHT AND PERMISSION NOTICE
Copyright (c) 1999,2000,2001 Compaq Computer Corporation
Copyright (c) 1999,2000,2001 Hewlett-Packard Company
Copyright (c) 1999,2000,2001 IBM Corporation
Copyright (c) 1999,2000,2001 Hummingbird Communications Ltd.
Copyright (c) 1999,2000,2001 Silicon Graphics, Inc.
Copyright (c) 1999,2000,2001 Sun Microsystems, Inc.
Copyright (c) 1999,2000,2001 The Open Group
All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

X Window System is a trademark of The Open Group.

OSF/1, OSF/Motif and Motif are registered trademarks, and OSF, the OSF logo, LBX, X Window System, and Xinerama are trademarks of the Open Group. All other trademarks and registered trademarks mentioned herein are the property of their respective owners.

The flexibility of the extended key segment support permits other implementations of Unicode support as well. The remainder of this section is focused on our ICU Unicode implementation. However, many of the topics, such as how an application can provide default key segment definitions will apply equally to any other future extended key segments.
How to Specify an ICU Unicode Key Segment

An ordinary c-tree Plus key segment is defined by a triplet: offset, length, and mode. In IFIL parlance, this is an ISEG structure. An extended key segment also uses this "standard" specification, but with two adjustments:

- The length specified in the triplet is the number of bytes that this segment will occupy in the key value stored in the index rather than the number of bytes of source data that will be used to generate the key segment. The source length will be part of the extended key segment definition discussed below.
- The key segment mode will contain a modifier that indicates the particular type of extended key segment. The only extended key segment at this time is UNCSEG, an ICU Unicode segment. An example of an ICU Unicode ISEG is:

  ISEG isegunc = \{8,24,REGSEG | UNCSEG\};

This ISEG specifies:
- The source data begins at an offset of eight bytes from the start of the record
- The key segment will be 24 bytes in length
- The segment type will be an ICU Unicode segment.

However, this ISEG definition does not specify the underlying data type (UTF8 or UFT16 for a Unicode segment), nor does it specify how many bytes of source data to use to construct the segment. The extended key segment definition specifies this additional information. Note that REGSEG implies no standard transformation, but the UNCSEG modifier specifies the particular type of extended segment.

In addition to an explicit definition of segment types such as REGSEG or INTSEG, c-tree Plus supports VARSEG and SCHSEG. VARSEG and SCHSEG use the same triplet, but the contents are interpreted somewhat differently.

- A VARSEG is a segment based on a field that falls in the variable length region of the file and therefore cannot be located by a simple offset value. The offset is interpreted as the number of fields in the variable length region to skip over. A zero implies the first variable length field is used.

To use an extended segment definition with VARSEG, simply modify the key segment mode as before:

VARSEG | UNCSEG

- A SCHSEG is a segment whose type is based on the data record field definitions stored in the DODA. When the mode is SCHSEG, the offset value is interpreted as a zero based index into the DODA. A value of zero implies using the first field definition to determine the type of key segment. It is permissible to use SCHSEG | UNCSEG, but not required if the offset value maps to an underlying data field that is one of the UTF16 Unicode types (CT_UNICODE, etc.).

However, if the underlying data is stored in a regular string field (e.g., CT_STRING), and the data is UTF8 encoded, c-tree Plus will not be able to automatically add the UNCSEG modifier. In this case, UNCSEG must be combined with SCHSEG:

SCHSEG | UNCSEG
## Extended Key Segment Definition

This section describes how to define an extended key segment.

**c-tree Plus**’s implementation of extended key segments allows a single extended key segment definition to be used by more than one actual key segment. For example, an application may make one call to `PutXtdKeySegmentDef` (discussed below) that applies to all of the extended segments used in the application. Therefore, some of the parameters specified in the definition optionally permit their particular values to be determined at run-time for each key segment.

Specify an extended key segment definition using the `ctKSEGDEF` structure presented in `ctport.h`:

```c
#define ctKSEGDLEN 32 /* length of desc string*/
typedef struct keysegdef {
    LONG kseg_stat; /* status (internal use)*/
    LONG kseg_vrsn; /* version info */
    LONG kseg_ssiz; /* source size */
    LONG kseg_type; /* segment type */
    LONG kseg_styp; /* source type */
    LONG kseg_comp; /* comparison options */
    LONG kseg_rsv1; /* future use */
    LONG kseg_rsv2; /* future use */
    TEXT kseg_desc[ctKSEGDLEN];
    /* text specification eg, locale string */
} ctKSEGDEF;
```

The **c-tree Plus** module `ctport.h` contains defines for all of the constants, beginning with `ctKSEG`, used to create an extended key segment definition. As extended key segments are currently implemented, the `kseg_stat` and the `kseg_vrsn` members are filled-in as needed by the extended key segment implementation itself. The `kseg_ssiz` member specifies the number of bytes of source data to use to derive the actual key segment. In addition to using a specific numeric value for the source size, `kseg_ssiz` may also be assigned either of these two "values:"

- **ctKSEG_SSIZ_COMPUTED** — The information about the underlying data field will be used to compute how much source data is available.
  - For fields without length specifiers (such as `CT_STRING` or `CT_UNICODE`) an appropriate version of `strlen()` will be used to determine data availability. However, this could be very inefficient if the field may hold very long strings since it is likely that only a small portion of the variable length field will actually contribute to the key segment. An alternative is to specify a fixed source size. If the variable data has less than this size, it will still be handled correctly.

- **ctKSEG_SSIZ_PROVIDED** — The call to create the key segment will provide the particular length of source data available. This option is typically used when an explicit call is made to `TransformXtdSegment` (discussed below).
For an ICU Unicode definition, the remaining structure members are specified as follows:

<table>
<thead>
<tr>
<th><strong>kseg_type</strong></th>
<th>Set to <strong>ctKSEG_TYPE_UNICODE</strong>.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>kseg_styp</strong></td>
<td>Specify the type of source data as follows:</td>
</tr>
<tr>
<td></td>
<td><strong>ctKSEG_STYP_UTF8</strong></td>
</tr>
<tr>
<td></td>
<td><strong>ctKSEG_STYP_UTF16</strong></td>
</tr>
<tr>
<td></td>
<td><strong>ctKSEG_STYP_PROVIDED</strong></td>
</tr>
<tr>
<td></td>
<td><strong>ctKSEG_STYP_PROVIDED</strong> means that the type of source data will either be specified in a direct call to <strong>TransformXtdSegment</strong> or it will be determined at run-time during ISAM key value construction. (Key value construction consists of one or both of assembling the key value from its component segments and performing transformations to generate a binary sort key.)</td>
</tr>
<tr>
<td></td>
<td><strong>IMPORTANT NOTE</strong>: <strong>ctKSEG_STYP_PROVIDED</strong> can be used during ISAM key value construction (e.g., with <strong>AddRecord</strong>) provided:</td>
</tr>
<tr>
<td></td>
<td>• A schema (DODA) exists for the associated data file</td>
</tr>
<tr>
<td></td>
<td>• The key segment mode is either <strong>VARSEG</strong> or <strong>SCHSEG</strong>.</td>
</tr>
<tr>
<td></td>
<td>In this case, if the data type is one of the conventional <strong>c-tree Plus</strong> string types (e.g., <strong>CT_STRING</strong>), the source data type is UTF8; if a Unicode string type is found (e.g., <strong>CT_UNICODE</strong>), then the source data type is UTF16. However, if the underlying data type does not fall into either of these categories, the data is treated as UTF16, and used as is.</td>
</tr>
<tr>
<td></td>
<td>Otherwise (i.e., if there is no DODA or the segment mode is not <strong>VARSEG</strong> or <strong>SCHSEG</strong>) error <strong>SSEG_ERR</strong> will be generated at the time of key value construction.</td>
</tr>
<tr>
<td><strong>kseg_desc</strong></td>
<td>Contains the ICU locale formed as an ordinary, null terminated ASCII string. The format specified by ICU is &quot;xx&quot;, &quot;xx_YY&quot;, or &quot;xx_YY_Variant&quot; where &quot;xx&quot; is the language as specified by ISO-639 (e.g., &quot;fr&quot; for French); &quot;YY&quot; is a country as specified by ISO-3166 (e.g., &quot;fr_CA&quot; for French language in Canada); and the &quot;Variant&quot; portion represents system dependent options.</td>
</tr>
<tr>
<td></td>
<td><strong>IMPORTANT NOTE</strong>: When ICU uses a locale to access collation rules, it attempts to get rules for the closest match to the locale specified in <strong>kseg_desc</strong>. By default, there is no restriction on how close the match of locales must be to be acceptable. You can restrict the use of alternative locales by including either <strong>ctKSEG_COMPU_FALLBACK_NOTOK</strong> or <strong>ctKSEG_COMPU_SYSDEFAULT_NOTOK</strong> as part of the bit map comprising <strong>kseg_comp</strong> discussed below. After a successful call to <strong>PutXtdKeySegmentDef</strong>, the <strong>GetXtdKeySegmentDef</strong> function can be used to determine the actual ICU locale used during collation.</td>
</tr>
<tr>
<td><strong>kseg_comp</strong></td>
<td>This member of the structure permits the full range of ICU collation options to be specified through a bit map. The details of these options are beyond the scope of this documentation. However the symbolic constants used to form the bit map are presented in the &quot;ICU Collation Option Overview&quot; section below.</td>
</tr>
</tbody>
</table>
ICU Collation Option Overview

The collation options can be grouped as follows: locale default control, collation strength, normalization, and special attributes. Locale default control affects the degree to which a default locale must be related to the requested locale. Collation strength determines how case, accents and other character modifiers affect the ordering of sort keys. Normalization affects how alternative variations of the "same" character (including its accents and other modifiers) are compared. The special attributes affect particular properties of the collation, which further modify the strength and normalization options. For example, a special attribute can be used to force lower case characters first or last in the collation.

If no locale default control option is made part of kseg_comp, there is no restriction on how close to the requested locale the effective locale must be. For example, if you request collation for the German language ("de"), you are likely to get a locale based on the system default (e.g., "en_US" in the United States). This is not a problem since it has been determined that the default rules work for the German language.

If ctKSEG_COMPU_SYSDEFAULT_NOTOK is used, then a request to use locale "xx_YY_Variant" will succeed as long as collation rules for "xx" are available. If ctKSEG_COMPU_FALLBACK_NOTOK is used, then rules for the particular locale with its optional country and variant modifiers must be available. Falling back from "xx_YY" to "xx" is not satisfactory. In the case of the "de" locale noted above, the segment definition would cause an error in the call to PutXtdKeySegmentDef if either of the "NOTOK" default restrictions are part of the definition.

At most one of the following collation strength options can be included in kseg_comp:

- ctKSEG_COMPU_S_PRIMARY
- ctKSEG_COMPU_S_SECONDARY
- ctKSEG_COMPU_S_TERTIARY
- ctKSEG_COMPU_S_QUATERNARY
- ctKSEG_COMPU_S_IDENTICAL
- ctKSEG_COMPU_S_DEFAULT

At most one of the following normalization options can be included in kseg_comp:

- ctKSEG_COMPU_N_NONE
- ctKSEG_COMPU_N_CAN_DECMP
- ctKSEG_COMPU_N_CMP_DECMP
- ctKSEG_COMPU_N_CAN_DECMP_CMP
- ctKSEG_COMPU_N_CMP_DECMP_CAN
- ctKSEG_COMPU_N_DEFAULT
One or more of the following special attributes can be included in kseg_comp. After each one of our symbolic constants is the equivalent ICU-attribute, attribute-value pair.

<table>
<thead>
<tr>
<th>ctKSEG_COMPU_A_FRENCH_ON</th>
<th>(UCOL_FRENCH_COLLATION, UCOL_ON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctKSEG_COMPU_A_FRENCH_OFF</td>
<td>(UCOL_FRENCH_COLLATION, UCOL_OFF)</td>
</tr>
<tr>
<td>ctKSEG_COMPU_A_CASE_ON</td>
<td>(UCOL_CASE_LEVEL, UCOL_ON)</td>
</tr>
<tr>
<td>ctKSEG_COMPU_A_CASE_OFF</td>
<td>(UCOL_CASE_LEVEL, UCOL_OFF)</td>
</tr>
<tr>
<td>ctKSEG_COMPU_A_DECOMP_ON</td>
<td>(UCOL_DECOMPOSITION_MODE, UCOL_ON)</td>
</tr>
<tr>
<td>ctKSEG_COMPU_A_DECOMP_OFF</td>
<td>(UCOL_DECOMPOSITION_MODE, UCOL_OFF)</td>
</tr>
<tr>
<td>ctKSEG_COMPU_A_SHIFTED</td>
<td>(UCOL_ALTERNATE_HANDLING, UCOL_SHIFTED)</td>
</tr>
<tr>
<td>ctKSEG_COMPU_A_NONIGNR</td>
<td>(UCOL_ALTERNATE_HANDLING, UCOL_NON_IGNORABLE)</td>
</tr>
<tr>
<td>ctKSEG_COMPU_A_LOWER</td>
<td>(UCOL_CASE_FIRST, UCOL_LOWER_FIRST)</td>
</tr>
<tr>
<td>ctKSEG_COMPU_A_UPPER</td>
<td>(UCOL_CASE_FIRST, UCOL_UPPER_FIRST)</td>
</tr>
<tr>
<td>ctKSEG_COMPU_A_HANGUL</td>
<td>(UCOL_NORMALIZATION_MODE, UCOL_ON_WITHOUT_HANGUL)</td>
</tr>
</tbody>
</table>

It is permissible to set kseg_comp to zero. A zero kseg_comp implies no restrictions on locale defaults, default collation strength, default normalization, and no special attributes.

For a complete treatment of all of these options, please refer to the ICU web site and the Unicode Consortium’s web site and publications.

### Extended Key Segment Definition Example

This example will simply show a completed ctKSEGDEF structure. The actual use of this structure is demonstrated in the example in the API section below.

```c
ctKSEGDEF ksgdef;

ksgdef.kseg_ssiz = 12; /* 12 bytes for the source */
ksgdef.kseg_type = ctKSEG_TYPE_UNICODE; /* ICU Unicode */
ksgdef.kseg_styp = ctKSEG_STYP_UTF16; /* UTF16 source data */
ksgdef.kseg_comp = ctKSEG_COMPU_A_LOWER; /* lower case sorts first */
strcpy(ksgdef.kseg_desc,"fr_CA"); /* French in Canada */
```
Extended Key Segment Default Hierarchy

If a key segment mode (the last member of the ISEG structure) includes a modifier for an extended key segment definition (e.g., REGSEG | UNCSEG), then the particular extended key segment definition to use for this segment is determined according to the following hierarchy. Use the definition specified for:

1. the segment
2. the index associated with the segment
3. the host index file containing the associated index
4. the data file associated with the index
5. the application default
6. if applicable, the server default

The PutXtdKeySegmentDef routine can specify extended key segment definitions for each of these six levels. To make this hierarchy more concrete, consider this example. Data file customer.dat has an index file named customer.idx. The index file customer.idx contains 3 indices: a customer number index, a customer name index, and a customer status index. The customer name index is comprised of one Unicode segment. If an extended key segment definition has been specified for this particular segment, then it is used. If not, and if an extended key segment definition has been specified for the customer name index, then it is used. If not, and if an extended key segment definition has been specified for the host index file (customer.idx), then it is used. If not, and if an extended key segment definition has been specified for the associated data file (customer.dat), then it is used. If not, and if an extended key segment definition has been specified for the entire application, then it is used. If not, and if the application is on a server, and if an extended key segment definition has been specified for the server, then it is used. If not, a USEG_ERR occurs: there is no extended key segment definition to use.

Except when an extended key segment definition has been specified for a particular key segment, the determination of which extended definition to use (as specified in the above hierarchy) is not determined until the first use of the key segment. By “first use” we mean either a reference to a key segment, say in a call to AddRecord or AddVRecord, after a file has just been created; or upon opening a file that contains extended key segment references that were not used after file creation and subsequent file closure. Upon this first use, if a default from one of levels 2 through 5 is used, then the particular definition is stored in the host index file so that the definition can travel with the physical file. (This automatic storage will not occur if the host index file is opened read only or has DISABLERES in its file mode.)

This hierarchy has been implemented to simplify the use of extended key segment definitions. One can easily imagine a Unicode dependent application that will only process Unicode key segments for one language, although the language may change from one site to another. By using a single call to PutXtdKeySegmentDef at the application level, the details of the Unicode segment including the locale can be specified at program startup. All extended key segments can then default to this
application definition. And since the definition will be added to the host index files, the index files created this way become self-sufficient.

It is important to note that except for a segment specific extended definition, there can be more than one extended key segment definition for each of the remaining five levels (2 through 6). But there can only be one extended key segment definition at each level for a particular type of segment. For example, there can be at most one ICU Unicode extended key segment definition at each level. (At this time we do not have any other type of extended key segment definition, but this is likely to change in time.)

Once an extended key segment definition has been specified at a particular level (for a particular type of segment), an attempt to specify another definition at the same level results in an error. This is in part because of the "first use" strategy noted above, and because one should not change a definition if key values already exist.

**Extended Key Segment API**

Three routines operate directly on extended key segments:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PutXtdKeySegmentDef</td>
<td>Defines an extended key segment for a Server, an application, a data file, an index file, or a particular index segment. Returns a handle for the definition if successful.</td>
</tr>
<tr>
<td>GetXtdKeySegmentDef</td>
<td>Retrieves (i.e., fills in the elements of a ctKSEGDEF structure for) the requested extended key segment definition. If successful, the return value is the handle associated with the definition.</td>
</tr>
<tr>
<td>TransformXtdSegment</td>
<td>Creates a binary sort key (segment) using an extended key segment definition. If successful, it returns the number of bytes used for the binary sort key.</td>
</tr>
</tbody>
</table>

All three functions return a negative value upon error, where the absolute value of the return value is the error code.

The `pkdef` parameter points to a definition to be created in a call to `PutXtdKeySegmentDef`. In a call to `GetXtdKeySegmentDef`, the structure pointed to by `pkdef` gets filled-in except that the `kseg_type` member of the structure should be set on input to the type of segment to be retrieved. For example, to retrieve an ICU Unicode definition, set the `kseg_type` member to `ctKSEG_TYPE_UNICODE`. 
Unicode Support  
Unicode Key Segments

`NINT PutXtdKeySegmentDef(NINT filno, NINT segno, pctKSEGDEF pkdef);`

<table>
<thead>
<tr>
<th>filno</th>
<th>segno</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctKSEGserver</td>
<td>ignored</td>
<td>Create server default definition</td>
</tr>
<tr>
<td>ctKSEGapplic</td>
<td>ignored</td>
<td>Create application default definition</td>
</tr>
<tr>
<td>datno</td>
<td>ignored</td>
<td>Create data file level definition</td>
</tr>
<tr>
<td>keyno</td>
<td>ctKSEGindex</td>
<td>Create index file level definition</td>
</tr>
<tr>
<td>keyno</td>
<td>0, 1, 2, ...</td>
<td>Create specific segment definition</td>
</tr>
</tbody>
</table>

`NINT GetXtdKeySegmentDef(NINT filno, NINT segno, pctKSEGDEF pkdef);`

<table>
<thead>
<tr>
<th>filno</th>
<th>segno</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctKSEGserver</td>
<td>ignored</td>
<td>Retrieve server default definition</td>
</tr>
<tr>
<td>ctKSEGapplic</td>
<td>ignored</td>
<td>Retrieve application default definition</td>
</tr>
<tr>
<td>ctKSEGhandle</td>
<td>handle</td>
<td>Retrieve definition associated with handle</td>
</tr>
<tr>
<td>datno</td>
<td>ignored</td>
<td>Retrieve data file level definition</td>
</tr>
<tr>
<td>keyno</td>
<td>ctKSEGindex</td>
<td>Retrieve index file level definition</td>
</tr>
<tr>
<td>keyno</td>
<td>0, 1, 2, ...</td>
<td>Retrieve particular segment definition</td>
</tr>
</tbody>
</table>

If `GetXtdKeySegmentDef` is called with `ctKSEGhandle` for the `filno` parameter and the handle value is passed in via the `segno` parameter (as shown in the third row in the above table), then the `kseg_type` member of the structure is ignored on input since the handle uniquely identifies the particular definition. On output, the `kseg_type` member will be set to the type of segment.

`NINT TransformXtdSegment(NINT seghnd, pVOID src, NINT srclen, pVOID dest, NINT destlen);`

<table>
<thead>
<tr>
<th>seghnd</th>
<th>Handle returned by <code>PutXtdKeySegmentDef</code> or <code>GetXtdKeySegmentDef</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>Pointer to data used to construct segment.</td>
</tr>
<tr>
<td>srclen</td>
<td>Size in bytes of the region pointed by <code>src</code>. However, <code>srclen</code> is ignored unless <code>kseg_ssiz</code> was set to <code>ctKSEG_SSIZ_PROVIDED</code>.</td>
</tr>
<tr>
<td>srcTyp</td>
<td><code>srcTyp</code> should be set to one of the <strong>c-tree Plus</strong> field types (e.g., <code>CT_STRING</code> or <code>CT_UNICODE</code>). However, <code>srcTyp</code> is ignored unless <code>kseg_styp</code> was set to <code>ctKSEG_STYP_PROVIDED</code>.</td>
</tr>
<tr>
<td>dest</td>
<td>Pointer to region in which binary sort key is constructed.</td>
</tr>
<tr>
<td>destlen</td>
<td>Size in bytes of the region pointed to by <code>dest</code>.</td>
</tr>
</tbody>
</table>
One of the main reasons to call `GetXtdKeySegmentDef` is to be able to examine the actual locale being used for the ICU collation routines. Most applications will not have a reason to call `TransformXtdSegment` unless the application needs to create a Unicode binary sort key outside of the normal ISAM processing.

### API Example

```c
#include "ctreep.h"

typedef struct datrec { /* offset: description */
    LONG delflg; /* 0: delete flag */
    LONG sernum; /* 4: serial number */
    TEXT idnum[8]; /* 8: ID number */
    TEXT utf8str[64]; /* 16: UTF8 string */
    TEXT zipcode[10]; /* 80: zipcode */
    TEXT code[6]; /* 90: codes */
} DATREC;

ISEG iseg[5] =
    { /* offset, length, mode */
        {80,10,REGSEG}, /* 0 */
        {16, 8,REGSEG | UNCSEG}, /* 1 */
        {16,24,REGSEG | UNCSEG}, /* 2 */
        {80, 5,REGSEG}, /* 3 */
        { 4, 4,SRLSEG} /* 4 */
    };

IIDX iidx[3] =
    {
        {22,0,1,0,0,2,iseg + 0},
        {33,0,1,0,0,2,iseg + 2},
        { 4,0,0,0,0,1,iseg + 4}
    };

IFIL ifil = {"datrec",-1,96,0,SHARE | TRNLOG, 3, 0,
    SHARE | TRNLOG,iidx};

main(int argc,char **argv)
{
    ctKSEGDEF sd;
    DATREC dr;
    NINT rc = 0, hnd;
    COUNT filno;

    if ((rc = INTISAM(100,16,16)))
        exit(rc);

    sd.kseg_ssiz = 12; /* use up to 12 byte */
    sd.kseg_type = ctKSEG_TYPE_UNICODE; /* ICU Unicode */
    sd.kseg_styp = ctKSEG_STYP_UTF8; /* UTF8 source data */
    sd.kseg_comp = ctKSEG_COMPU_S_TERTIARY |
        ctKSEG_COMPU_N_DEFAULT;
}
```
strcpy(sd.kseg_desc,"ar"); /* arabic */

if ((hnd = PutXtdKeySegmentDef(ctKSEGapplic,0,&sd)) < 0) {
    /* hnd holds the negative of the error code value */
    CLISAM();
    exit(-hnd);
}

/*
** else hnd holds the handle associated with the application-wide default for ICU Unicode extended key segments. Two of
** the segments specified in the ISEG array require ICU Unicode extended key segment definitions. If no other definitions
** are available upon the first use of the indices containing
** the UNCSEG modifiers, then the application default
** definition will be used.
*/

if ((rc = CREIFIL(&ifil))) {
    CLISAM();
    exit(rc);
}

/*
** else the data and indices have been successfully created. No
** extended key segment information has been added to either
** the data file or the index file at this point. If a call to
** PutXtdKeySegmentDef is now made that explicitly references
** the data file or index file (using ifil.tfilno or ifil.tfilno +
** keyno), then a special resource would be added upon
** successful completion of the PutXtdKeySegmentDef call.
*/

memset(&dr,0,sizeof(dr));

/*
** dr.delflg is now set to zero, and dr.sernum (now zero) will
** be filled-in by a call to ADDREC
*/

strcpy(dr.idnum,"1234567");
strcpy(dr.zipcode,"99999");
strcpy(dr.code,"YNM");

/*
** the ordinary ASCII fields are now set
*/

getUnicodeUTF8string(dr.utf8str,64);

/*
** this external routine will fill-in the multi-byte UTF8
** encoded Unicode string up to 64 bytes
*/
if (!TRANBEG(ctTRNLOG | ctENABLE) ||
    (rc = ADDREC(ifil.tfilno,&dr)) ||
    (rc = TRANEND(ctFREE))) {
/* could not successfully add data record */
    CLISAM();
    if (!rc)
        rc = uerr_cod;
    exit(rc);
}

/*================================================================**/
/* else successfully added data record. This first use of the          */
/* UNCSEGs will have caused the index file to be updated not         */
/* only with the key values, but the application default            */
/* extended key segment definition also will have been added to     */
/* the index.                                                       */
/*================================================================**/
    CLISAM();
    exit(0);
}

### New Error Codes

The three new functions all return negative values if an error occurs. Also, an ISAM operation might fail with a new error code if a problem arises attempting to use an extended key segment. A problem with an encoded file name might also produce a new error. The following table describes the new error codes related to extended key segments and Unicode file names.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>694</td>
<td>NUNC_ERR</td>
<td>Executable does not support ICU Unicode, but a UNCSEG modifier has been encountered.</td>
</tr>
<tr>
<td>700</td>
<td>OSEG_ERR</td>
<td>Could not process key segment definition. This could happen during a PutXtdKeySegmentDef call or when a file is opened that includes an extended key segment definition. Ordinarily, the CTSTATUS.FCS file will contain additional information about the problem.</td>
</tr>
<tr>
<td>701</td>
<td>CSEG_ERR</td>
<td>Could not process the kseg_comp options. This could occur if more than one of a set of mutually exclusive options are combined.</td>
</tr>
<tr>
<td>702</td>
<td>ASEG_ERR</td>
<td>An error occurred when attempting to process one of the special attribute options.</td>
</tr>
<tr>
<td>703</td>
<td>HSEG_ERR</td>
<td>Invalid key segment handle in a call to TransformXtdSegment or in a call to GetXtdKeySegmentDef when the ctKSEGHandle option is used and the segno parameter should be set to a valid extended key segment handle.</td>
</tr>
</tbody>
</table>
### Unicode Support

#### Unicode Key Segments

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>704</td>
<td>SSEG_ERR</td>
<td>No source type provided when <code>kseg_styp</code> has been set to <code>ctKSEG_STYP_PROVIDED</code>. If this error occurs, it is likely to occur during the first use (say with an <code>AddRecord</code> or <code>AddVRecord</code> or <code>OpenIFile</code>) of the extended key segment.</td>
</tr>
<tr>
<td>705</td>
<td>DSEG_ERR</td>
<td>An extended key segment definition already exists at the level implied by the <code>PutXtdKeySegmentDef</code> call.</td>
</tr>
<tr>
<td>706</td>
<td>NSEG_ERR</td>
<td>Zero bytes of binary sort key were generated. Possibly an all NULL source.</td>
</tr>
<tr>
<td>707</td>
<td>USEG_ERR</td>
<td>There is no extended key segment definition to use.</td>
</tr>
<tr>
<td>708</td>
<td>MBSP_ERR</td>
<td>Multibyte/Unicode file names are not supported.</td>
</tr>
<tr>
<td>709</td>
<td>MBNM_ERR</td>
<td>A badly formed multibyte/Unicode file name has been encountered.</td>
</tr>
<tr>
<td>710</td>
<td>MBFM_ERR</td>
<td>A multibyte/Unicode variant is not supported (e.g., UTF32)</td>
</tr>
</tbody>
</table>

The following table contains explanations of existing error codes used in this new context.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>LERR_ERR</td>
<td><code>PutXtdKeySegmentDef</code> called for a data or index file requires the file to be opened exclusively. (Remember, a just created file is in exclusive mode, regardless of the specified file mode, until it is closed and re-opened.)</td>
</tr>
<tr>
<td>437</td>
<td>DADR_ERR</td>
<td>NULL <code>pkdef</code> argument in <code>PutXtdKeySegmentDef</code> or <code>GetXtdKeySegmentDef</code>.</td>
</tr>
<tr>
<td>445</td>
<td>SDAT_ERR</td>
<td>No source data to create key segment.</td>
</tr>
<tr>
<td>446</td>
<td>BMOD_ERR</td>
<td>Improper <code>filno</code> or <code>segno</code> values in calls to <code>PutXtdKeySegmentDef</code> or <code>GetXtdKeySegmentDef</code>. <code>TransformXtdSegment</code> causes a BMOD_ERR if the handle references an extended key segment definition not supported by the executable.</td>
</tr>
<tr>
<td>589</td>
<td>LADM_ERR</td>
<td>Member of non-ADMIN group called <code>PutXtdKeySegmentDef</code> for a Server default (i.e., <code>ctKSEGserver</code>).</td>
</tr>
</tbody>
</table>
### Server Configuration Keywords For Unicode Segment Default

The server configuration file (or settings file or command line) can specify a server default for each type of extended segment definition supported. Each such default definition starts with a `XTDKSEG_SEG_TYPE` entry. For an ICU Unicode default, the entries begin with

```
XTDKSEG_SEG_TYPE UNICODE_ICU
```

This initial entry is then followed by the specifics of the default such as locale and other collation options. For example, the following configuration entries would define a default for the server with the same characteristics as the application default used in the above API example.

```
XTDKSEG_SEG_TYPE UNICODE_ICU
ICU_LOCALE "ar"
XTDKSEG_SRC_SIZE 12
XTDKSEG_SRC_TYPE UTF8
ICU_OPTION STRENGTH_TERTIARY
ICU_OPTION NORM_DEFAULT
```

The complete list of keywords and arguments is shown below:

<table>
<thead>
<tr>
<th>keywords</th>
<th>arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>XTDKSEG_SEG_TYPE</code></td>
<td>UNICODE_ICU</td>
</tr>
<tr>
<td><code>XTDKSEG_SRC_TYPE</code></td>
<td><code>PROVIDED</code></td>
</tr>
<tr>
<td></td>
<td><code>UTF8</code></td>
</tr>
<tr>
<td></td>
<td><code>UTF16</code></td>
</tr>
<tr>
<td><code>XTDKSEG_SRC_SIZE</code></td>
<td><code>PROVIDED</code></td>
</tr>
<tr>
<td></td>
<td><code>COMPUTED</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;numeric value&gt;</code></td>
</tr>
<tr>
<td><code>XTDKSEG_FAILED_DEFAULT_OK</code></td>
<td><code>YES</code> [server can still begin if server default encounters an error]</td>
</tr>
<tr>
<td></td>
<td><code>NO</code> [server cannot continue on error, which is the default behavior]</td>
</tr>
<tr>
<td><code>ICU_LOCALE</code></td>
<td><code>&lt;locale string in ICU form: xx_YY_Variant&gt; *</code></td>
</tr>
</tbody>
</table>
## Unicode Support

### Unicode Key Segments

<table>
<thead>
<tr>
<th>keywords</th>
<th>arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICU_OPTION</td>
<td>STRENGTH_PRIMARY, STRENGTH_SECONDARY, STRENGTH_TERTIARY, STRENGTH_QUATERNARY, STRENGTH_IDENTICAL, STRENGTH_DEFAULT, NORM_NONE, NORM_CAN_DECMP, NORM_CMP_DECMP, NORM_CAN_DECMP_CMP, NORM_CMP_DECMP_CAN, NORM_DEFAULT, LOCALE_SYSDEFAULT_NOTOK, LOCALE_FALLBACK_NOTOK, ATTR_FRENCH_ON, ATTR_FRENCH_OFF, ATTR_CASE_ON, ATTR_CASE_OFF, ATTR_DECOMP_ON, ATTR_DECOMP_OFF, ATTR_SHIFTED, ATTR_NONIGNR, ATTR_LOWER, ATTR_UPPER, ATTR_HANGUL</td>
</tr>
</tbody>
</table>

*Where "xx" is the language as specified by ISO-639 (e.g., "fr" for French); "YY" is a country as specified by ISO-3166 (e.g., 'fr_CA' for French language in Canada); and the "Variant" portion represents system dependent options.*

For all but ICU_OPTION, only one of the listed arguments can be specified for each keyword. For instance, it does not make sense to have both of these entries in the configuration file for one extended key segment default definition:

```
XTDKSEG_SRC_TYPE UTF16
XTDKSEG_SRC_TYPE UTF8
```

There may be many ICU_OPTION entries in a configuration file. Some combinations of entries do not make sense, and the behavior is not guaranteed if they are combined. For instance, using both of these entries is inappropriate:

```
ICU_OPTION ATTR_LOWER
ICU_OPTION ATTR_UPPER
```
5.5 Unicode File Names

Virtually everywhere an ordinary ASCII file name is used with c-tree Plus, a Unicode file name can be used. However, to use Unicode file names, the underlying OS platform must support file creation with Unicode file names. Also, c-tree Plus requires that the file name have a special 8 byte prefix that informs c-tree Plus about special file name encoding. Currently, on the WIN32 platform, c-tree Plus accepts the use of UTF8 and UTF16 encoded Unicode file names.

The function NINT ctMBprefix(pTEXT dp,NINT FnType) stores a proper prefix at dp of type FnType. dp is a pTEXT because the name may be encoded as a byte stream or a wide character array, and ctMBprefix does not assume that dp is aligned when used with UTF16. FnType may be ctFnTypeUTF8 or ctFnTypeUTF16. Both of these constants are defined in ctport.h. ctFnPrefixSize holds the size of the prefix in bytes. ctMBprefix returns NO_ERROR unless the FnType parameter is bad, in which case BMOD_ERR is returned.

In the following example, getUnicodeUTF8string is assumed to be a routine which fills in a UTF8 encoded string up to a maximum length. getUnicodeUTF16string performs in the same manner with 16 bit wide characters.

```c
COUNT datno8, datno16;
WCHAR utf16name[256];
TEXT utf8name[512];

cTMBprefix(utf8name,ctFnTypeUTF8);
getUnicodeUTF8string(utf8name + ctFnPrefixSize,512 –
cTFnPrefixSize);
datno8 = OPNRFIL(-1,utf8name,SHARED);

cTMBprefix((pTEXT) utf16name,ctFnTypeUTF16);
getUnicodeUTF16string(utf16name + ctFnPrefixSize / 2,
256 – ctFnPrefixSize / 2);
datno16 = OPNRFIL(-1,utf16name,SHARED);
```

**NOTE:** There is no provision for wide character encoded file name extents when using IFIL routines, such as CreateIFILXtd, that permit the default file name extents to be overridden. It is possible to use UTF8 encoded extents that will be properly merged with the root file names. However, there are only 8 bytes reserved for the extents, which effectively means at most 3 Unicode characters. Of course, it is possible to fully specify data and index names complete with extents if the dataextn and indexextn parameters to CreateIFILXtd point to blank strings (i.e., " "), and if the aidxnam index name pointers (in the IIDX structures) specify the index file names complete with their extents.

**Mirrored File Names**

The prefix marking a Unicode encoded file name is only applied to the beginning of the mirrored file name string, NOT after the name separator ‘|’ (vertical bar).
6. Advanced File Encryption

The c-tree Server supports encryption of data, index and transaction log files using a proprietary encryption algorithm. This technology provides the means to add an extra level of confidentiality to an application's data. Once encrypted, it becomes difficult for a casual user to "dump" or "inspect" the data.

FairCom designed its proprietary encryption algorithm for speed and efficiency, focusing on minimizing performance loss. This Standard File Encryption is not intended as a replacement for OS or other security systems, but as a supplement to existing security. Standard File Encryption is suitable for most needs. The Advanced File Encryption options, described below, offer additional alternatives.


The goal of Advanced File Encryption is to expand developer options and produce a suite of protocols that will protect the user data by what is loosely called strong encryption with a certain amount of performance overhead. The algorithms and protocols used are based on three primitives:

- Secure One-Way Hash Function (MD5)
- Block Ciphers (DES and AES)
- Pseudo-Random Number Generators

This chapter describes these Advance Encryption options:

- The “Using Advanced File Encryption” chapter describes how to implement the FairCom-defined encryption algorithms in your custom Server.
- The “Cryptographic Techniques” chapter describes in some detail each one of the above primitives.

6.1 Using Advanced File Encryption

Advanced File Encryption differs from Standard File Encryption in the following ways:

- Allows “stronger” encryption methods.
- Allows developer-defined encryption methods.
- Ciphers are specified in SetEncryption with the mod parameter.

Development

To prepare to use Advanced File Encryption,

- Build a Custom Server with the ctCAMOsdk #define added to ctree.mak/ctoptn.h.
- Build a client library with the ctCAMOsdk #define added to ctree.mak/ctoptn.h.
Server Defaults

When Advanced File Encryption is enabled, the Custom Server defaults to AES encryption with the CBC mode for internal encryption uses, such as log encryption. This is hard-coded in ctcryp.c, but can be changed to suit your needs.

Client Implementation

When Advanced File Encryption is enabled, the client application will function as with Standard File Encryption except that the mod parameter of SetEncryption is used to specify which cipher should be used.

To encrypt files, simply call SetEncryption before calling the function to create the file. The mod parameter should point to a text string containing one of the constants in the table below (e.g., “ctENCR” to use the default encryption method or “ctDES24” to use DES encoding with a 24 byte key.)

The possible mod values are defined in ctport.h:

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctRIJ16 ctRIJ24 ctRIJ32</td>
<td>Advanced Encryption Standard (AES) - Rijndael encryption algorithm implementation based on code made public by the Rijndael web page as an NIST AES finalist. For more information regarding this standard, refer to <a href="http://www.esat.kuleuven.ac.be/~rijmen/rijndael/">http://www.esat.kuleuven.ac.be/~rijmen/rijndael/</a>. According to the Rijndael web site: &quot;Rijndael is available for free. You can use it for whatever purposes you want, irrespective of whether it is accepted as AES or not. &quot;</td>
</tr>
<tr>
<td>ctBLF8 through ctBLF56</td>
<td>Blowfish encryption algorithm implementation based on code made public by Bruce Schneier of Counterpane Internet Security Inc. For more information regarding this standard, refer to <a href="http://www.counterpane.com/blowfish.html">www.counterpane.com/blowfish.html</a>. According to the Counterpane web site about Blowfish: &quot;Blowfish is unpatented and license-free, and is available free for all uses. &quot;</td>
</tr>
<tr>
<td>ctTWF16 ctTWF24 ctTWF32</td>
<td>Twofish encryption algorithm implementation based on code made public by Counterpane Internet Security Inc, as one of the NIST AES finalist. For more information regarding this standard, refer to <a href="http://www.counterpane.com/twofish.html">www.counterpane.com/twofish.html</a>. According to the Counterpane web site about Twofish: &quot;Twofish is unpatented, and the source code is uncopyrighted and license-free; it is free for all uses. &quot;</td>
</tr>
</tbody>
</table>

Server Configuration

Before starting the Server for the first time, the Administrator must use the ctpvyf utility to create the master password verification file.

Each time the Server starts, it will ask for the master password to allow it to open encrypted files.
ctcpf Utility – Create Master Password Verification File

ctcpf [filename [password]]

ctcpf creates the master password verification file. It accepts optional parameters: filename (the file name to create) and password (the master password). If the parameters are not given, ctcpf will prompt for the required information.

NOTE: At this time the Server looks for ctsrvr.pvf by default, so this file name should be specified.

Client Operation

As with Standard File Encryption, when using Advanced File Encryption SetEncryption is only required to create encrypted files. Any standard client can access files encrypted by a Custom Server configured for Advanced File Encryption. Encryption and decryption happen behind the scenes, invisible to the client application.

To protect files further, add a file password to each file. Only applications with the file password can open the file. See Chapter 12 of the c-tree Plus Programmer’s Reference Guide for more details.

6.2 Cryptographic techniques

The reason to use modern cryptography is to achieve the following:

Data confidentiality – Messages must be decrypted in order for information to be understood. This is accomplished by the use of block ciphers.

Data integrity – Messages may need to be protected against forgery or tampering. This is accomplished by the use of secure one-way hash functions.

Authentication – The message originator may need to be authenticated to decide who has access to information. This is accomplished by the use of one-way hash functions and block ciphers.

Digital signature – The system may need to be protected against unauthorized modification and forgery. This is accomplished by the use of one-way hash functions and block ciphers.

Secure One-Way Hash Functions

A hash function is a function that takes a variable length input string, called a pre-image, and converts it to a fixed length (generally smaller) output string called a hash value.

A one-way hash function is a hash function that works in one direction: it is easy to compute a hash value from pre-image, but it is hard to generate a pre-image that hashes to a particular value. A good one-way hash function is also collision-free: It is hard to generate two pre-images with the same hash value.
The hash function is public, as there is no secrecy to the process. The security of a one-way hash function is its "one-wayness". The output is not dependent on the input in any discernible way. A single bit change in the pre-image changes, on the average, half of the bits in the hash value. Given a hash value, it is computationally unfeasible to find a pre-image that hashes to that value.

MD5 Hash Function

MD5 message-digest algorithm created by RSA, takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest.

The MD5 algorithm is designed to be quite fast on 32-bit machines. In addition, the MD5 algorithm does not require any large substitution tables; the algorithm can be coded quite compactly.

We will use the following formula for one-way hash functions: \( d = H(p) \), where: \( d \) is the result of the hash function, also called the digest, \( p \) is the pre-image string, and \( H() \) is the MD5 hash function.

We will also use the term "\( d = H(p_1, p_2, \ldots, p_n) \)" to indicate the hashing of several pre-image strings to generate one digest.

Block Ciphers

The majority of the encryption algorithms in use today are block algorithms, which operate on one chunk (generally 64 or 128 bits) of data at a time.

We use the following formula to indicate encryption: \( c = E_k(p) \), where \( c \) is the cipher text after encryption, \( k \) is the encryption key, and \( p \) is the plain text.

We use the following formula to indicate decryption: \( p = D_k(c) \), where \( p \) is the plain text after decryption, \( k \) is the decryption key, and \( c \) is the cipher text.

Data Encryption Standard – DES

Designed at IBM during the 1970s and officially adopted as the NIST standard encryption algorithm for unclassified data in 1976, DES has become the bastion of the cryptography market. However, DES has since become outdated, its long reign as official NIST algorithm ending in 1997. Though DES accepts a 64-bit key, the key setup routines effectively discard 8 bits, giving DES a 56-bit effective key length. DES remains in wide use. During the design of DES, the NSA provided secret S-Boxes. After differential cryptanalysis had been discovered outside the closed fortress of the NSA, it was revealed that the DES S-boxes were designed to be resistant against differential cryptanalysis. DES is becoming weaker and weaker over time; modern computing power is fast approaching the computational horsepower needed to easily crack DES.

DES was designed to be implemented only in hardware, and is therefore extremely slow in software. A recent successful effort [http://www.frii.com/~rcv/deschall.html]{http://www.frii.com/~rcv/deschall.html} to crack DES took several thousand
computers several months. The Electronic Freedom Foundation has sponsored the development of a crypto chip named "Deep Crack" that can process 88 billion DES keys per second and has successfully cracked 56 bit DES in less than 3 days.

**Triple DES with Two keys – 2DES**

This idea, proposed by W. Tuchman, operates on a block three times with two keys. This works by first encrypting the block with the first key, then decrypting with the second key and encrypting again with the first key. This method is also called 2DES. This method gives DES an actual key length of 112 bits (16 bytes).

To encrypt: \( C = E_{k_3}(D_{k_2}(E_{k_1}(P))) \)

To decrypt, we reverse the process: \( P = D_{k_1}(E_{k_2}(D_{k_3}(C))) \)

This method has been adopted to improve the security of the DES algorithm in X9.17 and ISO 8732 standards. This method improves the security of DES at the expense of speed.

**Triple DES with Three keys – 3DES**

An even more secure way to use triple encryption is to use three different DES keys. This method gives DES an actual key length of 168 bits (24 bytes). Again the improved security is gained at the expense of speed. This method is also called 3DES.

To encrypt: \( C = E_{k_3}(D_{k_2}(E_{k_1}(P))) \)

To decrypt: \( P = D_{k_1}(E_{k_2}(D_{k_3}(C))) \)

**Advanced Encryption Standard – AES (Rijndael)**

Rijndael is a block cipher, designed by Joan Daemen and Vincent Rijmen and was the chosen algorithm for the AES (Advanced Encryption Standard) by NIST in October 2000.

The cipher has a variable block length and key length. Options are included to specify keys with a length of 128, 192, or 256 bits to encrypt blocks with a length of 128, 192 or 256 bits (all nine combinations of key length and block length are possible). Both block length and key length can be extended very easily to multiples of 32 bits.

The AES was adopted by NIST with a block size of 128 bits and key lengths of 128, 192 and 256 bits. AES is considered to be very secure and probably will replace DES in most cryptographic applications.

**Electronic CodeBook Mode – ECB**

A block of plain text encrypts into a block of cipher text. This is the easiest mode to work with. Each plain text block is encrypted independently. The problem with ECB mode is that if a cryptanalyst has the plain text and cipher text for several messages, he can start to compile a code book without knowing the key. This vulnerability is
greatest at the beginning and end of messages, where well-defined headers and footers may contain information about the sender, receiver, date and so on.

The plain text stream length must be a multiple of the encryption algorithm block size.

**Cipher Block Chaining Mode – CBC**

In CBC mode, the plain text block is XORed with the previous cipher text block before it is encrypted; the resulting cipher text is stored in a feedback register with the same size of the encryption algorithm block size. Before the next plain text is encrypted, it is XORed with the feedback register.

\[ c_1 = E_k(p_1 \oplus c_{i-1}) \]

Decryption is just as straightforward. A cipher text block is decrypted normally and also saved in a feedback register. After the next block is decrypted, it is XORed with the results of the feedback register.

\[ p_i = c_{i-1} \oplus D_k(c_i) \]

The plain text stream length must be a multiple of the encryption algorithm block size.

**Output-FeedBack Mode – OFB**

Output-feedback mode is a method of running a block cipher as a synchronous stream cipher. On both the encryption and decryption sides, the block algorithm is used in its encryption mode. This is sometimes called internal feedback, because the feedback mechanism is independent of both the plain text and the cipher text streams.

Encryption:

\[ s_i = E_k(s_{i-1}) \]

\[ c_i = p_i \oplus s_i \]

Decryption:

\[ s_i = E_k(s_{i-1}) \]

\[ c_i = c_{i-1} \oplus s_i \]

The plain text stream may be of any length.

**Pseudo-Random Number Generator**

Computers are deterministic beasts. Data goes in one end, completely predictable operations occur inside, and different information comes out the other end. Feed the same data in two different occasions and the same information comes out on both of them. Put the same data in on two different computers and the same information is generated. A computer can only be in a finite number of states (a large finite number) and that means that any random number generator on a computer is, by definition, periodic. Anything that is periodic is predictable. A true random number generator requires some random input and a computer can’t provide that.
The best a computer can produce is a pseudo-random sequence generator that looks random and with a period that should be long enough.

A good pseudo-random sequence generator must have the following qualities:

- It looks random. This means that it passes all the statistical test of randomness.
- It is unpredictable. It must be computationally infeasible to predict the next random bit, given complete knowledge of the algorithm or hardware generating the sequence and all the previous bits in the sequence.

### Random Bit Stream Generator

The **c-tree Server SDK** uses the ANSI X9.17 standard to generate a stream of pseudo-random bits:

\[
R_i = E_k(E_k(T_i) \oplus V_i) \\
V_{i+1} = E_k(E_k(T_i) \oplus R_i)
\]

Where:
- \( R_i \) is the generated random bits
- \( T_i \) is the timestamp, i.e., the hash of the system time
- \( k \) is the random seed

The formula above shows that the quality of the pseudo-random number generator depends mostly on the quality of the random seed. The section below describes strategies for generating cryptographically secure random seeds for generating pseudo-random numbers.

### Random Seed

It is not easy to collect random seed data from a computer. It is almost impossible to do it in a way that is portable for the most common operating systems in use today.

For Microsoft Windows systems (Win32 system based on Windows 9x, Windows ME, Windows NT and Windows 2000), the **c-tree Server SDK** uses Microsoft Crypto API.

The C code fragment below illustrates this approach:

```c
#define SEED_SIZE 32

int RandomSeed(unsigned char* buffer)
{
    HCRYPTPROV FProv = 0;
    HCRYPTKEY FKey = 0;
    DWORD FSize;

    if (!CryptAcquireContext(&FProv,NULL,NULL,PROV_RSA_FULL,0))
    {
        return 1;
    }
```
if (!CryptGenKey(FProv, CALG_RC4, 0, &FKey))
{
    return 2;
}
memset(buffer, 0, SEED_SIZE);
FSize = SEED_SIZE;
if (!CryptEncrypt(FKey, 0, TRUE, 0, buffer, &FSize, SEED_SIZE))
{
    return 3;
}
if (FKey) CryptDestroyKey(FKey);
if (FProv) CryptReleaseContext(FProv, 0);
return 0;

On Linux systems, the seed value is obtained by reading the /dev/urandom device, which provides cryptographically secure pseudo-random numbers.

On Unix systems, without the /dev/urandom device, the seed value is generated by hashing the following data:

- output of function call gettimeofday()
- output of function call clock()
- contents of file /var/log/messages or /var/adm/messages
- output of command "netstat -a"
- output of command "netstat -i"
- output of command "ps -eaxlw" or "ps -eafl" on SysV systems
- output of command "dmesg"
- output of command "uptime"
- output of command "openssl rand 32" if available
- output of function call gettimeofday()

The output of the hash function is the seed for the random number generator.
7. Portability Enhancements

New platforms (Windows CE and QNX RTP) and platform-specific enhancements for Mac, Windows, Novell, HP-UX, SCO OpenServer, NetBSD, FreeBSD, and Solaris (Intel) are detailed below.

7.1 New Platform Support

FairCom continues to lead the industry in the number of platforms that we commercially support! In fact, we take pride in the fact that we support everything from embedded operating systems such as QNX and LynxOS to desktop operating systems like Windows ME and Mac OS, to server operating systems like Solaris and HP-UX.

We are continuing that tradition in this new release with added support for Windows CE and QNX RTOS v6. Don’t think we’re stopping there—we’ve already planned two additional ports for the next quarter!

Windows CE Port – Standalone Models

FairCom is pleased to announce the support for Windows CE. As embedded solutions continue to demand the effectiveness of c-tree Plus, we are happy to bring formal support for the Windows CE operating environment.

With project-level support for the Microsoft embedded Visual C++ compiler, FairCom allows scalable support across all versions of Microsoft Windows, from Windows CE through Windows 2000 and XP, with our eyes open to future development. With a consistent API across all platforms, developers can easily adapt c-tree Plus applications from other Windows platforms for use on Windows CE.

The compact and efficient c-tree Plus engine is perfectly suited to the embedded environment, but flexible enough to scale to the most powerful Server environment.

Compiling for Windows CE:

1) Install Microsoft eMbedded Visual C++ compiler on a Windows NT or 2000 system.
2) Install c-tree Plus V7.11.
3) Start a command prompt.
4) Run WCE\ex86.bat to set up the Windows CE development environment.
5) Change to mtree directory and create a makefile using mtmake with platform #14 (Windows CE).
6) Change to mak.xxx directory (where xxx is replaced with sgl for single-user, fpg for multi-user, or mfp for multi-threaded) and open creestd.vcw with eMbedded Visual Studio (EVC).
7) Select your Active WCE Configuration.
8) Select your Active Configuration.
9) Press F7 to build the sample (ctree\samples\cetest\*.*)
10) When the build is complete, press F5 to run the sample.
QNX RTP Platform Added

FairCom has ported c-tree Plus and the c-tree Server to the latest operating system from QNX Software Systems, QNX RTOS v6. The QNX realtime platform (RTP) is built on the QNX Neutrino realtime OS, one of the most popular and advanced real-time operating systems on the market.

FairCom has been a development partner with QNX Software Systems for many years and our solutions have been deployed together in many different industries including the industrial, medical-equipment, telephony, and automotive fields.

QNX currently offers its real-time operating system to developers at no charge from its web site at www.qnx.com.

7.2 Enhanced Platform Support

In addition to the two new platforms that we are now supporting with this release, we’ve enhanced our support for a number of additional platforms.

Updates for Mac Developers

2001 was a good year for Mac developers. The release of Max OS X marked a significant milestone for Apple, and revitalized the Mac developer community.

FairCom included support for Mac OS X in our V6.10 release, before Mac OS X was commercially released. So in case you missed the news in our last release and in case you missed our booth at Apple’s World Wide Developers Conference, we do formally support Mac OS X.

Carbon Support

With the release of Mac OS X, Apple introduced the Carbon API. Carbon is the set of programming interfaces derived from earlier Mac OS APIs that can run in Mac OS X. Some of these APIs have been modified or extended to take advantage of Mac OS X features such as preemptive multitasking and protected memory.

c-tree Plus Standalone models have been formally “carbonized” to support building applications for both the Mac OS 9 and Mac OS X platforms. Updated CodeWarrior projects include the proper settings to control this new support.
Mac OS X Cocoa Interface for c-tree Server

In anticipation of the commercial release of Apple’s recent Mac OS X, we gave the c-tree Server a face-lift on this platform. A new graphical front-end interface, using Apple’s Cocoa development tools, has been added to the Server on Mac OS X.
Shared Libraries for Mac OS X

The current make files address implementing shared libraries for Max OS X. This feature allows multiple applications to share a single c-tree Plus library, minimizing executable size.

Standalone Multi-threaded Model Enhanced

Support for Multi-Threaded multi-user applications (FPUTFGET) has been improved for the Apple Mac platform. A number of code cleanup and adjustments were involved in verifying this support and enhance performance and stability.

Server Shutdown Prompt Adjusted

The c-tree Server for Apple Mac shutdown screen, which indicates if any users are still logged onto the Server, now accepts keystrokes to select either CANCEL or YES. Prior to this change, the mouse was required. This allows a keystroke-input stream to automate shutting down the Server.

Symantec 7 Projects updated

At the request of a customer, the old Symantec 7 projects were resurrected in order to satisfy a legacy need. The code modifications were minimal which is a testament to the well-coded heart of c-tree Plus.

Updates for Windows Developers

c-tree Server for Windows continues to provide a powerful solution for developers deploying on Windows platforms, combining the power and flexibility of c-tree Plus in a low cost, low maintenance database server.

If you are still deploying using our standalone models, we encourage you to try our development servers that are included on the c-tree Plus V7.11 CD. We know you and your customers will appreciate the added features that our server solutions can bring to your application.

Dynamic Message Monitor and Function Monitor

Menu options have been added to the c-tree Server for Windows in order to dynamically turn on/off both the function monitor as well as the console Message windows. This adjustment will make it easy for developers to activate the function monitor without having to stop and then restart the Server.

Server Shutdown as Windows NT/2000 Service

The c-tree Server can be run as a service on Windows NT/2000. If the properties are configured to allow interaction and the user clicks the Control-Shutdown icon, the Server now prompts the user to Continue or Cancel before stopping. If the user selects Continue, the service shuts down properly and the Server comes down cleanly without further interaction. In the past, the Server and service were simply shut down without a prompt.
Additional Platform-Specific Updates

We’ve made a few additional platform-specific changes that may affect the platforms you are utilizing.

NetBSD, FreeBSD, and Solaris (Intel) allow 4 GB Files

The default for the NetBSD, FreeBSD, and Solaris (Intel) platforms has been updated to support up to 4 GB Standard files. Support for Extended files larger than 4 GB is offered with c-tree Plus Professional V7.

HP-UX 64 bit Compile Option Added

The HP9000 options have been updated in the m-tree make system in order to support 64 bit compiling on the HP-UX 11 operating system. Now developers may choose between 32 bit or 64 bit compile options. Shared libraries are supported in both cases.

Shared Library Support for HP-UX 11 and SCO OpenServer 5

The m-tree system now includes shared library options for HP-UX 11 and SCO OpenServer 5. This feature allows multiple applications to share a single c-tree Plus library, minimizing executable size.

New GUI NLM Rename Utility

Developers who want to run more than one instance of the c-tree Server for Novell are already familiar with the rennlm console utility. Now the new GUI version of this utility is available, as shown below.
Simply provide the original file name from which the utility can generate a new NLM (by browsing the file folders, if desired):

Then provide a new name and click Finish:

The new NLM is placed in the same directory as the original. See “Running Multiple c-tree Servers on One Machine” in the c-tree Server Administrator’s Guide.
c-tree Server SDK for NetWare NLM: MetroWerks CodeWarrior

The option to compile and link a c-tree Server for NetWare (ctsrvr.nlm) has been added to the m-tree make system main menu. When this option is selected, the following information is presented:

--- FairCom NLM Server ---

You have chosen to generate a FairCom NLM Server for NetWare. This option requires the MetroWerks CodeWarrior v6 or better with CodeWarrior's Add-On for NetWare. In addition, Novell's NWSDK must also be present, and referenced by the NOVELLNDDK environment variable. Your CodeWarrior Project will be placed in:

<yourdir>\Build\NetWare.Nlm\ctsrvr

Press 'ENTER' then launch the Code Warrior IDE and open the 'ctsrvr' Project. Select Project/Make (F7) to build the FairCom NLM Server.

Simply follow these directions to build ctsrvr.nlm.
8. Improved Deleted Space Management

A new key segment mode allows enhanced deleted space management, reverse-physical-order record reads, and record counts for variable-length files.

The RECBYT segment mode specifies an index segment containing the character ‘D’ in the first byte. The rest of the segment is padded with NULL bytes, however, there is no reason to use a segment length greater than one.

To take advantage of the new variable-length (VLEN) space management capabilities, create a RECBYT index, which is an index using:

- Only a RECBYT segment
- A segment length of one
- Allows duplicates. Since a duplicate key adds the record position as part of the actual key value, this results in a 5-byte (9-byte for huge files) duplicate key with a single byte for the segment (‘D’ for active data records).

Use a RECBYT index for backward physical traversal of fixed length files containing resources or any variable length files. Use NbrOfKeyEntries on this index for an exact count of variable-length records in the data file.

8.1 Improved Deleted Space Management

When a RECBYT index exists for a given variable-length data file, etree Plus uses the index to improve space management for the data file. When a variable-length record is deleted, if the space following the deleted record is already deleted, the two spaces merge into one deleted space. A RECBYT index adds the capability to check and possibly merge the previous adjacent deleted record space.
8.2 Example

The following example adds a RECBYT index to the ISEG/IIDX structures in \texttt{ctixmg.c}:

\begin{verbatim}
ISEG segments[] = {
    {0,15,5}, /* First index key segments */
    {4,5,2}, /* Second index key segments */
    {0,4,1},
    {4,9,2}, /* Third index key segments */
    {0,9,5},
    {0,1,RECBYT} /* RECBYT key segment with a
                        segment length of one. Segment offset is ignored */
};

IIDX ndxs[] = {
    {24, 4,1,0,0,0,1,&segments[2]},
    {22,12,1,1,32,2,&segments[3],NULL,"vcusti.2"},
    {5, /* 1 byte segment length plus 4 byte offset */
     0,1, /* Allow Duplicates */
     0,0,1,&segments[5]}
};
\end{verbatim}
9. Server Enhancements

The following enhancements are detailed fully in this chapter:

- Improved performance on multi-CPU systems due to the alignment of high-use shared objects on cache memory boundaries.
- Enhanced monitor windows for the **c-tree Server**.
- Improved detection of dropped or inactive client connections by the **c-tree Server**.
- The space allocated to files deleted from superfiles can now be reclaimed and a standalone utility can change the sector size of existing superfiles.
- File-specific Encryption – Allows a specific encryption key for each file.
- Other Miscellaneous Enhancements.

9.1 Multi-CPU Performance

Many internal improvements have been made to enhance the performance of the **c-tree Server** under multi-CPU systems. Structures have been sized, re-arranged, and aligned to minimize cache refreshing.

A cache-line is the smallest amount of memory a processor will retrieve and store in its highest speed cache. Typical cache-line sizes are 16, 32 or 64 bytes. On multiple CPU systems, when a memory word is updated, any other processor with a cache-line containing the updated word must eliminate or refresh the cache-line. When the other processors are not actually using the particular word that was updated, this is called false-sharing.

Many of the Server's high use objects (e.g., semaphores, mutexes, transaction related arrays) are now structured and/or allocated in a manner sensitive to the false sharing problem. This modification minimizes performance loss from false-sharing. Semaphores and mutexes are allocated to ensure each semaphore exists in its own cache-line and an update in one semaphore's state by a CPU will not cause other CPUs to synchronize cache.

The cache-line size defaults to 16 and may be overridden with a configuration entry of the form: `CACHE_LINE <size>`. Set this value to the appropriate value for your CPU, as described above, to maximize performance.

9.2 c-tree Server Detects Dropped Clients

The **c-tree Server** normally recognizes when a client disconnects. However, the Server relies on a chain of events controlled by the operating system in order to recognize the disconnection. The client computer must notify the Server host computer that the connection has been dropped. For example: When a user closes an application, the socket is closed by the operating system, which sends a message to the Server host machine. However, if the network connection is temporarily interrupted or if the client machine is powered down suddenly, this message is not
sent and the Server host machine can’t recognize that the client connection has dropped.

With the `COMPATIBILITY TCP/IP_CHECK_DEAD_CLIENTS` keyword in the Server configuration, the c-tree Server detects when a TCP/IP client has dropped. Every 120 seconds, the client connection socket is rechecked to ensure that it is still a valid communications channel. If this connection is found to be invalid, the Server will terminate this connection.

This functionality is not supported on the c-tree Servers for Mac, OS/2, or early Linux (kernel less than version 2.036) at this time.

### 9.3 Server Detects Inactive Clients

A new keyword `SESSION_TIMEOUT` has been added that instructs the Server to remove TCP/IP connections after a specified number of seconds have elapsed without any activity. The server checks connections periodically (every 5 minutes on most platforms) to determine whether the `SESSION_TIMEOUT` interval has elapsed and, if so, whether the connection should be terminated for inactivity. This allows clients with no activity to be disconnected from the Server automatically.

```
SESSION_TIMEOUT <seconds>
```

The theoretical maximum number of seconds is over 2 billion, so any practical values are supported. For example, a value of 1800 would remove the connection after 30 minutes. This new keyword is off by default.

This feature has been tested on Windows, Novell, Linux, Mac OS 9, and Mac OS X.

### 9.4 Superfile Space Reclamation

It is now possible for a Superfile controlled by the c-tree Server to reclaim the space from deleted member files. The basic approach is to move the space from a deleted superfile member to the host's space management index, then make this space available when a member data file's reusable space is empty.

The space reclamation is performed by a dedicated background thread. A permanent queue permits the space reclamation to be interrupted at Server shutdown, and resumed when the server restarts. The Server file `D0000001.FCS` holds queue entries for the space reclamation of deleted superfile members.

Add `COMPATIBILITY NO_SPCMGT_QUEUE` to the Server configuration to disable the space reclamation queue.
9.5 Server Files Transaction Dependent

In V7, the creation of the administrative files FAIRCOM.FCS, SYSLOGDT.FCS, and SYSLOGIX.FCS is transaction dependent. This should eliminate problems where a Server coming up for the first time is not permitted to complete the startup sequence. Now, these files will either be:
- recreated from scratch if their creation sequence was interrupted; or
- will be okay if their creation sequence completed.

There is actually one separate creation sequence for the FAIRCOM.FCS file and its superfile members, and another for the SYSLOGxy pair of files.

This use of transaction dependent creates demonstrates an important strategy: if there is a collection of files which should be created in an all or nothing manner, their creation can be done within a transaction which guarantees the atomicity of the creation sequence. Further, not only can the creation be atomic, but if certain initialization of the files is required, then this can also be included within the same transaction to ensure its atomicity.

9.6 File-specific Encryption Control

In systems supporting file encryption, control over whether or not to encrypt a file is now available on a file-by-file basis.

If file encryption has been enabled by a call to SetEncryption, an individual file can disable encryption by or’ing ctNOENCYP into the x8mode member of the XCREb1k structure.

Whether or not SetEncryption has been called, setting the filkey member of the XCREb1k structure to a non-zero value causes the file to be encrypted using the value of the filkey member as the encryption key.

9.7 Diagnostic Enhancements

The following features and enhancements provide additional options for developers troubleshooting their applications.

Function Monitor Enhancements

The FUNCTION_MONITOR window now displays the file name in addition to the User ID and function name to allow Server Administrators to monitor activity for specific files.

The optional FUNCTION_MONITOR log (created when a file name is a parameter to the FUNCTION_MONITOR keyword in ctsrvr.cfg) now includes the c-tree Plus return value (error number) for every c-tree Plus function called by the client. This option gives the ability to execute a client application, as is, and inspect in this log the return values of all c-tree Plus function calls made by the client application.
**DIAGNOSTICS LOWL_FILE_IO Keyword**

The **DIAGNOSTICS LOWL_FILE_IO** Server keyword logs low-level system errors into the Server status file, `CTSTATUS.FCS`. Although client applications have access to system errors through `sysiocod`, it is useful to see these errors logged on the Server side.

For example: An end-user has problems opening a file. The end-user copied the data file from a CD-ROM onto the hard disk leaving the file marked “read-only”. When the user attempted to open these files, open errors were generated within the application. Adding the **DIAGNOSTICS LOWL_FILE_IO** keyword directed the Server to log the “not authorized” operating system error to `CTSTATUS.FCS`. This pointed the user to the “read-only” issue.

**DIAGNOSTICS WRITETHRU Warning**

The **c-tree Server** writes a warning message to the `CTSTATUS.FCS` file if it detects a file that is not under transaction control and does not have `WRITETHRU` defined. This warning is to notify the developer that the file is being maintained in a vulnerable mode. Because of the overhead of writing this message to the log, and because FairCom does allow this “dangerous-cached-buffered-type” mode, the warning message is only issued once, for the first file detected. In other words, it simply tells the developer that there was “at-least-one” vulnerable file.

To help developers detect the file names for all vulnerable files, a **DIAGNOSTICS WRITETHRU** keyword has been added. By placing the following in your `ctsrvr.cfg`, all file names will be listed in the `CTSTATUS.FCS` file:

```
DIAGNOSTICS WRITETHRU
```

**TRAP_COMM Utility Added**

The **DIAGNOSTICS TRAP_COMM** Server keyword allows developers to record the communication traffic coming in to a Server and play it back with the `cttrap` utility.

**cttrap – Communications Trap Playback utility**

```
cttrap <TrapCommLogFileName> [ <ServerName> ]
```

`cttrap` is a multithreaded client application that plays back a `TRAP_COMM` log file. Whenever a Multithreaded Client library is created, cttrap will also be generated. The default `TRAP_COMM` file name is `TRAPCOMM.FCS`.

**DIAGNOSTICS TRAP_COMM**

When activated, the **DIAGNOSTICS TRAP_COMM** keyword instructs the **c-tree Server** to log incoming communications packets to `TRAPCOMM.FCS` prior to execution. This log can be played back using the `cttrap` utility and a debug Server to observe the results of the client requests, allowing the developer to exactly duplicate and repeat client activities.
The trap file, *TRAPCOMM.FCS*, is created in the Server directory by default. To prepend a path onto the trap file name (say to route it to a separate disk), add an entry of the form `DIAGNOSTIC_STR <trap file path>`. For example, if `DIAGNOSTIC_STR /bigdisk/` were in the configuration file, then the trap file would be `/bigdisk/TRAPCOMM.FCS`.

To collect a proper TRAP_COMM session:

1. Ensure the data files are in a “clean” state, i.e., the error condition to be trapped does not yet exist with these files.
2. Keep a backup of the data files in this “clean” state. These files will be used again to play back the TRAP_COMM file.
3. Execute the Server with the TRAP_COMM keyword in place.
4. Reproduce the error.
5. Use the saved data files and the TRAP_COMM file for diagnostic work. In most cases, these will need to go to FairCom for evaluation.

### Debug Node I/O Diagnostics Enhanced

A number of internal improvements provide more detailed diagnostics information related to Index Node I/O. Periodic reports of a rare un-reproducible `BNOD_ERR(69)` error justified this effort where we applied internal diagnostics designed to report internal states when this error occurs.

Adding the `#define DBGnodeIO 10000`, where 10000 is the size of an internal buffer, before building the *c-tree Server* causes the Server to maintain a circular buffer of information that will offer FairCom insights into this problem. If the 69 error is encountered, the Server writes the buffer to the *CTSTATUS.FCS* file.

### Diagnostics for the 7626 Error

FairCom added the following diagnostics to the *c-tree Server* to debug the 7626 error:

1. **System Event Log:** Each time the Server detects a key value being deleted that was added within the scope of the same transaction, the operation will be recorded in the Server system event log. This provides a picture of the frequency of this type of operation within the PDMA application. Activate this logging with the `SYSLOG ANL` keyword in the Server Configuration File, *ctsrvr.cfg*. This introduces minimal overhead, assuming that this operation occurs infrequently.

2. **Key and File:** The precise key value and related file information are written to disk if the 7626 error should occur. This provides the data to be used in conjunction with the other diagnostics to follow.

3. **Recent Abort Node List Activity:** Internally, a circular buffer has been introduced that logs the most recent 15,000 operations against the Abort Node List. On the occurrence of the 7626 error, this log is dumped to the file
4. **Dump of Abort Node List**: On the occurrence of the 7626 error, the entire Abort Node List is logged to the *ABNODDMP.FCS* file. This occurs automatically without the need for additional diagnostics keywords.

### 9.8 Logon Timeout Enhanced

The Server *LOGON_FAIL_TIME* keyword now supports a value of –1, which indicates that there should be a permanent "log-on block" placed on a user. To help put this in context, the *LOGON_FAIL_LIMIT* keyword specifies the number of "strike-outs" or "failed logon attempts" that the Server allows before "blocking" additional logons for a specific user. The *LOGON_FAIL_TIME* keyword specifies the time that a user should be prevented from logging on after the User ID has been invalidated. This change allows this invalidation period to be forever/permanent or at least until the Server ADMIN intervenes.

### 9.9 Change log and dynamic dump buffer sizes

The buffer sizes for the transaction log and dynamic dump have been changed to 64K. This value was not used in V6 because of segmented architectures and I/O limitations. The change has a potential for improved performance.

### 9.10 Coordinate deletes and dynamic dumps

The *c-tree Server* has been changed to better coordinate file deletes and dynamic dumps. Before this modification, the Server would only stop a requested delete on the file actively being dumped. A file could be deleted after the dump started but before the actual dump of the particular file took place.

Now, a file created with the *ctTRANDEP* file mode bit (in the *x8mode* member of the XCREblk extended file create block) that is deleted after the effective dump time will be sure to exist at the time this particular file is dumped. This is accomplished by making the delete wait until the file is dumped.

### 9.11 Client-side Broadcast Functionality Enhanced

**GetServerInfoXtd**, declared as `GetServerInfoXtd(pTEXT buffer, NINT bufsiz, UCOUNT port, LONG sec)`, allows two new parameters.

`GetServerInfoXtd` listens for a broadcasting *c-tree Server* on TCP/IP port *port* for *sec* seconds before timing out. The *port* value should match the TCP/IP port set in the Server *BROADCAST_PORT* keyword. A negative *sec* defaults to 90 seconds. If a broadcast is received, up to *bufsiz* bytes are stored at *buffer*. 

---

*ABNODLST.FCS* file. This capability is activated with the *DIAGNOSTICS ABORT_NODE_LIST* keyword in the Server Configuration File, *ctsrvr.cfg*. 

---

*ABNODDMP.FCS* file.
9.12 Support PAGE_SIZE Up To 65536

Adding #define ctFeatPAG64K at compile time enables support for a maximum PAGE_SIZE of 65536. Without this modification, the maximum effective PAGE_SIZE is 32768. Before this modification, no check about the suitability of a requested PAGE_SIZE was made. Now the PAGE_SIZE is limited to a maximum of 32768 or 65536 depending on whether ctFeatPAG64K has been defined. Such a large PAGE_SIZE would only be of importance if very long key lengths are in use and/or the disk storage I/O would be optimized with this PAGE_SIZE.

9.13 CHECKPOINT_INTERVAL adjustments

The CHECKPOINT_INTERVAL Server keyword has been adjusted:
- Increased default maximum checkpoint interval from 1MB to 2MB
- Added additional CTSTATUS.FCS output at startup concerning log file size and checkpoint interval

9.14 SystemLog additions

This release adds support for deleted files to be entered into the system log files. Placing SYSLOG DELETE_FILE in the Server configuration causes three types of evclass ctSYSLOGdelfil events to be logged in SYSLOGDT.FCS:

<table>
<thead>
<tr>
<th>Event</th>
<th>Variable Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctSYSLOGdelfRSTR</td>
<td>12 byte File ID, Original Name, Copy Name</td>
<td>Restore of deleted file.</td>
</tr>
<tr>
<td>ctSYSLOGdelfTRAN</td>
<td>12 byte File ID, Original Name, Copy Name</td>
<td>Transaction-dependent delete</td>
</tr>
<tr>
<td>ctSYSLOGdelfNOTRAN</td>
<td>12 byte File ID, Original Name</td>
<td>Non-transaction-dependent delete</td>
</tr>
</tbody>
</table>
10. c-tree Plus Enhancements

The following enhancements are detailed fully in this chapter:

- Transaction dependent file creates and deletes. This feature allows file creation and deletion under transaction control, including full rollback and roll-forward capability. This can be critical for systems requiring a specific set of files to be created or destroyed as a group.
- Improvements in key segment handling allow any field type to be supported in the variable-length portion of the file.
- Adaptive node splits – To help improve the "density" of the b+ tree structures built when key values are added in sorted order, the recent key insertion history of each buffer (which holds the contents of an index node) is automatically maintained and used to determine how to split a full node.
- File-specific Disk-full Checks – Determine disk status on a per-file basis.
- Other Miscellaneous Enhancements.

10.1 Transaction Dependent Creates and Deletes

A major new transaction-processing feature has been added to c-tree Plus V7. This feature, known as Transaction Dependent Creates/Deletes, supports the creation and deletion of files under transaction control. In prior releases, the physical creation and deletion of files from disk was handled outside the scope of transaction control. In other words, prior releases did not support, as an example, "aborting a create", or "rolling back a delete".

File creates and deletes may now be part of a transaction and are subject to being undone in case of crashes, savepoint restores, and aborts. This is accomplished with the extended file mode \texttt{ctTRANDEP}. A file with \texttt{ctTRANDEP} in its extended file mode is assumed to utilize transaction dependent creates and deletes and must be created within a transaction. There is no special function call for transaction dependent creates or deletes.

If a different client attempts to open or create a file pending delete, \texttt{DPND_ERR(643)} is returned to signify that opens/creates must await the commit or abort of the delete.

To utilize a transaction dependent delete with a file not created with \texttt{ctTRANDEP}, it will be necessary to update the file header to include the \texttt{ctTRANDEP} bit in the \texttt{x8mode} member of the extended header.

To support the rollback of a transaction dependent file delete, which requires V7 to maintain a copy of the old file after the delete is committed, use the \texttt{ctRSTRDEL} extended file mode. \texttt{ctRSTRDEL} automatically implies \texttt{ctTRANDEP}, but the reverse is not true. When only \texttt{ctTRANDEP} is used, the overhead of storing copies of an old file after its delete is committed is avoided.

A transaction dependent create is only supported by the new V7 create routines, and requires an extended file mode with \texttt{ctTRANDEP} and/or \texttt{ctRSTRDEL} included. To
perform a transaction dependent create of a Standard c-tree Plus file, call one of the new create routines with \texttt{ctNO\_XHDRS} and either \texttt{ctTRANDEP} or \texttt{ctRSTRDEL} included in the extended file mode. Including \texttt{ctNO\_XHDRS} in the extended file mode causes the new create routines to create a Standard (V6 compatible) file. If \texttt{ctNO\_XHDRS} conflicts with other extended file mode bits, then the create will fail with \texttt{XCRE\_ERR(669)}.

To convert an existing Standard file to support transaction dependent deletes, use V7 calls of the form:

\begin{verbatim}
    UpdateHeader(\textit{filno},ctTRANDEP,ctXFL\_ONhdr);
    or
    UpdateHeader(\textit{filno},ctRSTRDEL,ctXFL\_ONhdr);
\end{verbatim}

However, these \texttt{UpdateHeader} calls cannot convert a Standard file to an Extended file.

- **Dynamic Dumps** - A file with \texttt{ctTRANDEP} in its extended file mode that is deleted after the effective dump time will be sure to exist at the time this particular file is dumped. This accomplished by making the delete wait until the file is dumped.

- **PermIIndex8** and **TempIIndexXtd8** support \texttt{ctTRANDEP} creates. Without \texttt{ctTRANDEP} creates, these routines cannot be called within a transaction. With \texttt{ctTRANDEP} creates, they **must** be called within a transaction.

### 10.2 Index Any Field Type in a Variable-Length Record

While you’ve always been able to store any kind of information, including binary blobs, in the variable-length portion of a record, new functionality allows indexing on any field of any type in the variable-length portion of a record. This requires a record schema (DODA) in the data file, which defines the fields in the record.

Schema based segment types, from \texttt{SCHSEG(12)} to \texttt{UVSCHSEG(15)}, are required for all segments when fixed-length fields are included in the variable-length portion.

For example, the following IFIL, DODA, and record structures result in two indices, one on the \texttt{NumericData} field and 20 bytes of the \texttt{TextName} field, and one on the \texttt{OtherNumber} and \texttt{FixedText} fields. All the fields are in the variable-length portion of the record.

\begin{verbatim}
ISEG sampleseg[] = {
    {1, /* The segment offset refers to schema field
          offset 1, the second field: NumericData. */
        2, 15}, /* segment type is 15, UVSCHSEG, a schema type */
    {3,20,15},
    {2,4,15},
    {0,9,15}};
IIDX sampleidx[] = {
    {22,0,0,0,0,2,&sampleseg[0]},
    {13,0,0,0,0,2,&sampleseg[2]}};
\end{verbatim}
IFIL sampleifil = {"sample",0,0,0,2,0,0,&sampleidx}; /* reclen=0, no fixed portion. All fields in VLEN */

DATOBJ sampledoda[] = {
{"FixedText", NULL, CT_FSTRING, 10},
{"NumericData", NULL, CT_INT2},
{"OtherNumber", NULL, CT_INT4},
{"TextName", NULL, CT_STRING, 500}
};

struct RECORD {
TEXT FixedText[10];
COUNT NumericData;
LONG OtherNumber;
TEXT TextName[500];
} dataRecord;

Note: All fields in the variable-length region must packed as opposed to aligned. No padding bytes are allowed between fields.

This capability is not supported by Conditional Indices.

In the past, the variable-length portion of the record could contain any kind of information, but only variable-length string fields could be indexed, and then only if all the preceding fields in the variable-length portion were variable-length string fields. Use this method with conditional indices or in a file without a DODA.

10.3 Adaptive Index Node Splits

To help improve the "density" of the b+ tree structures built when key values are added in sorted order, the recent key insertion history of each buffer (which holds the contents of an index node) is automatically maintained and used to determine how to split a full node. Denser b+ trees result in fewer reads, enhancing performance.

The position in the node of the last key value added to the node and the number of ordered adds are stored for each buffer. If a sufficient number of ordered adds has occurred at the time the node overflows, the node is split preferentially as though INCADD or DECADD had been specified in the low level AddKey call.
10.4 Disk Full Enhancements

The Disk Full feature introduced in c-tree Plus V6.10 has been enhanced in V7 with the ability to turn on/off disk full checking on a file-by-file basis and the ability to define the disk full threshold on a file or volume basis.

Three levels of control over disk full checks exist:
- The DISK_FULL_LIMIT provides checking on all files.
- The volume-specific limit overrides the system-wide limit.
- The file-specific check overrides the system-wide and volume-specific checks.

File-by-file Disk Full Checks

To disable disk full checks for a particular file:
- Set the ctNOSPACHK bit in the x8mode member of the xCREblk structure.
- Create the file using the 8 byte extended creates (e.g., CreateDataFileXtd8).

To enable file-by-file checking with a file-specific disk full threshold:
- Set the dskful member of the xCREblk structure to the desired limit in bytes.
- Create the file using the 8 byte extended creates (e.g., CreateDataFileXtd8).

If extending the size of the file would leave less than the threshold specified in dskful, then the write operation causing the file extension fails, returning SAVL_ERR(583).

Volume Disk Full Checks

The c-tree Server keyword DISK_FULL_VOLUME takes as its argument a concatenated volume name and limit. A path separator must occur between the volume name and the threshold limit, which may be zero.

In Unix this is in the form /name/<limit>. The following example places a disk full threshold of one million bytes on the volume /home:
   DISK_FULL_VOLUME /home/1000000

In DOS, Windows, and OS2 the form of the argument is <DRIVE>:\<limit>. The following example places a 1MB threshold on drive E:
   DISK_FULL_VOLUME e:\1048576
10.5 API Enhancements

GetSymbolicNames Function Enhanced

The GetSymbolicNames function returns a name (string) associated with a file, such as the file name, mirror name, or the file owner’s UserID. Two new modes, DEXTNAM and IEXTNAME, allow GetSymbolicNames to return the data file extension and the index file extension, respectively. These options are useful in making calls, such as RebuildIFileXtd, that require the data and index file extensions as arguments, especially when operating without knowledge of the original IFIL structure.

Superfile Member EXCLUSIVE open – Standalone Multi-user ONLY

To lock superfile members when the host is opened in shared mode requires a special case of the ctLOCK_EXTENDED technique. It is enabled with ctSUPER_MBRLOCK and may be applied to either ctLOCK_DIRECT or ctLOCK_EXTENDED. By default, prior to this change, the ctSUPER_MBRLOCK was set in the FPUTFGET mode. Now, this option is inactive by default, and therefore now defaults to not supporting SuperFile Member file exclusive locks without modification. This improves performance, but prevents a Superfile member from being opened EXCLUSIVE when the superfile host is open SHARED. Compile with #define ctSUPER_MBRLOCK to allow applications to perform EXCLUSIVE opens in this manner.

MAXLEV Default Increased from 11 to 15

The internal default #define MAXLEV 11 has been changed to #define MAXLEV 15. MAXLEV defines the maximum levels c-tree Plus allows in an index binary tree.

Previously, c-tree Plus terminated with a 240 error return when MAXLEV was exceeded, since an index would only get this deep because of an inefficient definition.

Now, the default has been increased and c-tree Plus was modified to return BIDX_ERR(527) when MAXLEV is exceeded instead of terminating.

Max Key Length - MAXLEN increased from 255 to 1024

The maximum supported size for key lengths has been enhanced to support keys up to 1024 bytes long. The prior maximum length of an index entry (key) was 255. V7 now offers this larger capacity, which will be of use to developers working with large key values.

NOTE: Although it is possible to compress keys with a total length up to MAXLEN (default 1024 bytes), only the first 255 and/or the trailing 255 bytes are compressed. The internal logic uses a one-byte length value to indicate the number of characters compressed.
DoBatch BAT_KEYS Option Skips Missing Key Values

The behavior of the DoBatch call was changed when using the BAT_KEYS option. DoBatch now returns records found for each and every key given. A customer reported unexpected behavior using the previous version of DoBatch when passing in a set of key values for input, the BAT_KEYS option. If one of the keys in the set did not exist, the batch terminated without retrieving any other values. This is not a common problem since the set of keys passed in is frequently based on keys known to exist (e.g., from a previous DoBatch request).

cTThrd Direct Queue Read/Write

In an effort to streamline the ctThrd message queue system and allow very high-speed queue calls, two new functions were added to the ctThrd API. ctThrdQueueWriteDirect takes the same arguments as a regular queue write, ctThrdQueueWrite, except that the queue entry now includes a direct reference to the message address. The contents are not copied into a buffer allocated by the queue write call. ctThrdQueueReadDirect provides the address and length of the next message to be read but does not copy the message contents. The return values for both functions are an error indicator.

See the function descriptions for additional details.

cMAXSORTBUF placed in ctoptn.h now overrides CTSORTBUF

A new #define cMAXSORTBUF has been added in order to allow the user to override the default buffer size used by the c-tree Plus rebuild. #define cMAXSORTBUF determined the amount of memory used to buffer key sorts when rebuilding the indices associated with a data file. A larger sort buffer, in most cases, will result in a quicker rebuild.

By default, this value is 16000. On many occasions, FairCom’s technical support team has found it necessary to advise the customer to increase this value. Now, by adding the #define cMAXSORTBUF to the ctoptn.h/ctree.mak, a user may adjust this value without the need to change the core code.

Example:

#define cMAXSORTBUF 64000

DropIndex Function Added

A new function, DropIndex, has been added to the c-tree Plus API. DropIndex permanently removes an index from an associated data file, including updating the IFIL resource. This is the reverse of the existing Permlndex function.

Note that dropping or adding indices can only be reversed when the file is transaction dependent (TRANDEP).

See the function description for additional information.
10.6 Utility Enhancements

Superfile Compact Utility Adjusts Sector Size

The superfile compact utility has been enhanced to allow the user to provide a sector size for the superfile. Now ctscmp can be used not only to compact superfiles but also to change their sector size. This change allows Administrators who created a FAIRCOM.FCS using one PAGE_SIZE setting to restart the Server with a new PAGE_SIZE setting but use the current FAIRCOM.FCS.

The command-line usage is now:

```
ctscmp [Y] [new_sect_size]
```

where the optional argument Y compacts the file without prompting to confirm, and the optional argument new_sect_size is the sector size of the resulting file.

New IFIL-based Rebuild Utility – ctrbldif

A new rebuild utility using the IFIL definitions stored in the header of a file has been added to c-tree Plus:

```
ctrbldif DataFileName [<UserId>] [UserPassword] [ServerName] [-purge] [-<sectors>]
```

ctrbldif reads the IFIL structure from DataFileName and calls RebuildIFileXtd to rebuild DataFileName and its associated indices.

UserID, UserPassword, and ServerName are only needed for client versions of this utility. FairCom recommends building this utility as a Single-user Standalone application.

-purge indicates that duplicate records should be purged.

-<sectors> is the sector size to use. The sector parameter is especially useful if you need to adjust a file’s PAGE_SIZE to match the c-tree Server.

New IFIL-based Compact Utility – ctcmpcif

```
ctcmpcif DataFileName [<UserId>] [UserPassword] [ServerName] [-purge] [-<sectors>]
```

cctcmpcif reads the IFIL structure from DataFileName and calls CompactIFileXtd and RebuildIFileXtd to compact and rebuild DataFileName and its associated indices. If ctcmpc cannot extract the IFIL from the target file, it will ask for the name of another copy of the file from which to extract the IFIL information.

UserID, UserPassword, and ServerName are only needed for client versions of this utility. FairCom recommends building this utility as a Single-user Standalone application.

-purge indicates that duplicate records should be purged.

-<sectors> is the sector size to use. The sector parameter is especially useful if you need to adjust a file’s PAGE_SIZE to match the c-tree Server.
11. c-tree Driver Enhancements

This chapter describes the enhancements made to the c-tree Drivers since the V6.10 c-tree Plus release. Unless a specific driver is mentioned, each change applies to all c-tree Drivers.

11.1 Performance Enhancements

Reduce driver instance switching overhead

The ODBC Driver now tracks the current session pointer using a static variable to determine the current c-tree state to avoid extra function calls.

Remove DOSFLUSH logic in c-tree V4 Driver

It was determined that DOSFLUSH is not required for the c-tree V4 ODBC Driver, so this option has been turned off. Testing shows that this change provided a significant performance gain for add/delete/update operations.

Only read fixed-length portion of variable-length records

If a query only involves fields in the fixed-length part of a variable-length record, the ODBC Driver now bypasses the reading of the variable-length part of the record. This can enhance performance, particularly when accessing data over a network with large variable-length records.

11.2 New Features

Huge and Segmented Files

The c-tree Drivers use c-tree Plus V7 libraries, allowing full access to Extended and Standard files, including Huge and Segmented file support. At this point the Drivers create only Standard files, but they can read and update Extended files.

Support for DSN-less ODBC Connections

The c-tree Drivers set several Registry keys. Use the ConfigDSN() function to programmatically configure the c-tree ODBC Driver Data Source. Most c-tree Driver subkeys correspond to configuration entries shown in the table below. The other entries in this section are only adjustable in the Registry. All the subkeys below are in the key `\HKEY_CURRENT_USER\Software\ODBC\ODBC.INI\ <dname>`, where <dname> defaults to one of the following:

- FairCom c-tree Driver for the c-tree V4 driver
- FairCom 32bit Driver for the 32bit c-tree Plus driver
- FairCom 16bit Driver for the 16bit c-tree Plus driver
Registry subkey | Configuration option
---|---
Alignment | Alignment
AllowUpdates | Allow Updates
CaseSensitive | Case-insensitive string Comparison
ColumnsOrderedBy | Column Order
DataBuffers | Data Buffer Size
DataDictionary | Script Name
DBQ | Data Dictionary Path
DebugCtree | c-tree Plus Debug
DebugIndex | Debug Indexes
Description | Description
Files | Number of Files
ForceClose | Not a configuration option
GuestLogin | Logon to Server as Guest
IndexBuffers | Index Buffer Size
KanjiConvert | Not a configuration option
MaxColSupport | Table’s max number of columns
OEMtoANSI | OEM to ANSI Data Conversion
Protocol | Driver Type
Sectors | Sector Size
ServerName | Server Name
SpecialTypes | Special Data Type Conversion DLL Name
StringDataPadding | Data Padding
StringKeyPadding | Key Padding

It is also possible to use these keywords in a connection string that uses a DSN. The effect is to override the particular settings that are specified. For example: “DSN=FairCom 32bit Driver;Alignment=4;” uses the settings from the registry for the data source ‘FairCom 32bit Driver’ and forces the alignment setting to 4. **Note:** Two registry settings are not supported in the command string: ForceClose and KanjiConvert.

## Adding a Shareable DSN

The Registry feature allows the creation and use of shareable file DSN’s by the 32bit c-tree ODBC Driver. The Driver creates unshareable file DSN’s automatically when creating user or system DSN’s.
To create a shareable file DSN:

1) Start the 32bit ODBC Administrator.
2) Select the “File DSN” tab.
3) Click the “Add…” button.
4) Select the ODBC driver for which you wish to create a file DSN (either “FairCom 32bit ODBC Driver” or “FairCom c-tree Driver”).
5) Click the “Advanced…” button and enter the desired driver-specific keywords (from the list above). Both DRIVER= (driver name) and DBQ= (data dictionary path) are required. For example:

   DRIVER={FairCom 32bit ODBC Driver}
   DBQ=c:\faircom\odbc\32bit

6) Click the “OK” button.
7) Click the “Next” button.
8) Enter the name of your new file DSN.
9) Click the “Next” button.
10) Click the “Finish” button.

If after you follow the above steps, the ODBC administrator returns the error “A connection could not be made using the file data source parameters entered. Save non-verified file DSN?”, you probably omitted some needed parameters or specified invalid parameter values (see step 5 above). Click “Cancel” and start over at step 1.

After a file DSN is successfully created in this manner, it can’t be configured from the ODBC Administrator, but it is a plain text file that can be edited to change its settings. This type of file DSN is known as a shareable file DSN – if you arrange for it to be shared by multiple machines (each with the c-tree ODBC Driver installed), it will serve as an ODBC data source that is identically configured on all of these machines.

**ODBC Driver Supports 4095 byte keys**

In order to allow the ODBC driver to access indexes with large keys, the ODBC Driver was adjusted to support keys up to 4095 bytes.

**Guest Login Setting Configurable on a Per-DSN Basis**

The function that indicates which features the ODBC Driver supports derives the DSN name so that it can look in the proper registry subkey for the DSN that is being used and retrieve the GuestLogin setting.

If using a DSN-less connection, the guest login setting defaults to Yes, without examining the registry. Previously, the default data source name (e.g., FairCom 32bit Driver) was checked. Also, the driver never prompts for username and password in multi-user mode, regardless of the guest login setting.
Driver supports DebugLog option in connection string

The ODBC Driver now supports the use of the `DebugLog` option in the connection string. This option indicates whether driver error messages should be displayed in dialog boxes or logged to `ctodbc.log`. The driver recognizes the following string values for `DebugLog`:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| Both  | Display error in dialog box and log to `ctodbc.log`.
| Screen| Display error in dialog box only. |
| File  | Log to `ctodbc.log` only. |
| None  | Do not log or display error messages. |

Notify User of Data Read Error

If the ODBC Driver is unable to retrieve a record, it now displays a dialog box indicating the error when the “c-tree Plus Debug” option is enabled. Previously, the driver returned a status to the SQL engine indicating that no more records are available without notifying the user that there was an error reading the record. This change makes the behavior consistent with the driver’s logic used to retrieve a record by key value.

Always Refresh Data Dictionary Contents on Connect

Previously, the ODBC driver only read the data dictionary script (FAIRCOM.DB) and added entries to the data dictionary (CTSYSCAT.FCS) the first time a process connected to the driver. Subsequent connections by the same process simply read the current contents of the data dictionary, without examining the data dictionary script. Now the driver always refreshes the contents of the data dictionary during each connection. The reason for changing this behavior is to ensure that the data dictionary is updated to the state of the data dictionary script for each connection. Prior to this change, it was possible for the data dictionary contents and data dictionary script contents to be out of sync after connecting (noticed when developing an application in the Visual Basic IDE).

Driver assumes byte-aligned fields in variable-length part of record

The c-tree Plus V7 code assumes that fields in the variable-length portion of the record are byte-aligned. The ODBC driver code, however, assumed that such fields are aligned according to the alignment value specified in the data file’s schema map (or overridden by the ODBC driver settings). For consistency, the ODBC driver code now assumes byte-aligned fields in the variable-length portion of the data record.

Driver does not use special delimiter for fixed-length fields

A customer observed that the ODBC driver uses the schema field delimiter to delimit both fixed- and variable-length string fields. However, this delimiter should only
apply to variable-length string fields. The driver now delimits only variable-length string fields with the schema field delimiter.

**Specify maximum rows per query in Driver connection string**

A customer requested the ability to set the default maximum number of rows returned per query by specifying a keyword in the driver connection string. The driver now supports a connection option named MaxRows. The syntax is `MaxRows=n`, where `n` is the default maximum rows returned per query, which can be overridden by calling `SQLSetStmtAttr(SQL_ATR_MAX_ROWS)`.

The evaluation driver is still limited to returning at most 100 records per query, regardless of the use of the MaxRows option.

### 11.3 Fixes

#### Serious Issues

**Data truncated errors adding records for Memo fields having lengths over 64K. V3.11(0529)**

When creating a MEMO field whose length is greater than 65535, the driver’s SQL engine was truncating this length to a two-byte value (potentially producing an inappropriately small value), which caused errors on adding records. This problem was avoided by applying a change to the SQL engine.

**Driver may truncate long string fields by writing null terminator at incorrect offset. V3.11(0430)**

Previously, the ODBC driver had two problems when writing long string data. When appending data to a string field, the ODBC driver used an incorrect offset for the position of the null terminator, possibly causing the data to appear to be truncated. It also computed an incorrect offset for the position of the null terminator, causing the same problem. This problem was observed when attempting to read OLE objects stored using Microsoft Access with the c-tree ODBC Driver.

#### Installation

**ODBC driver InstallShield installer may fail to overwrite files that are in use without warning. V3.11(0430)**

Previously, if files that the ODBC driver installer was trying to install were in use, it created temporary files in the Windows directory, and installed these on the next reboot. However, the installer didn’t notify the user of this or prompt the user to reboot. To solve this problem, the “Potentially locked” setting has been turned off in the InstallShield projects. Now the user will get an error if the file is in use during installation.
Application-Specific Fixes / Crystal Reports

Add registry settings to prevent “Driver not capable” error when using Crystal Reports Visual Linking Expert. V3.11(0126)

A customer reported a “Driver not capable” error, which prevented him from linking his tables when using the Crystal Reports 8 Visual Linking Expert. The problem was resolved by changing two of Crystal Reports’ database options: selecting File | Options…, and on the “Database” tab, turning on “Use Indexes or Server for Speed” and turning off “Auto-SmartLinking”. These two options correspond to specific values in the registry, which the driver now sets at installation time.

Add registry settings to ensure correct join syntax when using Crystal Reports Visual Linking Expert. V3.11(0307)

Even after the registry changes were applied in Build(0126) of the driver, Crystal Reports still sometimes generated incorrect SQL syntax. FairCom researched the registry settings that are required in order to ensure that Crystal Reports generates the correct SQL syntax for joins and modified the installation accordingly.

Application-Specific Fixes / Visual Basic

Long fixed-length string fields appear blank in Visual Basic. V3.11(0907)

OT_GETIT() now returns the full defined field length for a fixed-length field if the specified destination length is zero. Prior to this change, OT_GETIT() returned the length of the data in the field. The result was a data truncated warning, causing Visual Basic to display the field as empty.

Application-Specific Fixes / Access

“Invalid buffer size” error with Access 2000 and long fixed-length string fields. V3.11(0719)

For compatibility reasons, the ODBC driver returns fixed-length strings padded with spaces, regardless of how they are padded on disk. If a long fixed-length string field contains a value that on disk has many trailing null bytes and the driver was asked to return the field in chunks, the driver subtracted the current offset in the field from the field’s actual length, producing a negative value and leading to an “Invalid buffer size” error, or a string value returned that was not padded with blanks to the full defined field length. This situation was resolved by computing the proper field position length value based on the type of field (fixed-length or variable-length).

FairCom also adjusted the string compare logic so that a string with trailing spaces is considered the same as a string without trailing spaces (e.g., ‘abc’ == ‘abc’).
“Dynamic parameter type mismatch” error occurs when deleting records using Microsoft Access. V3.11(0316)

A customer reported the error “Dynamic parameter #3 type mismatch” when deleting records using Microsoft Access. FairCom has resolved the problem.

Pad fixed-length strings with spaces to solve #Deleted problem with Access 2000. V3.10(1012)

When viewing data in a linked ODBC table with Access 2000, if fixed-length string data was not padded with spaces to the full field length, Access 2000 marked the records as #Deleted. The standard practice is for ODBC drivers to pad fixed-length string data to the full field length before passing the data to ODBC applications. The driver now does this, preventing Access 2000 from displaying the #Deleted message in this situation. This change was also applied to the Driver SDK source code.

This change affects the behavior of the LIKE operator on fixed-length strings that are not padded with spaces on disk. Because the SQL engine now sees the string with trailing spaces, a search for an expression such as the first line below should be changed to the second line to account for the trailing spaces.

- SELECT name FROM mytable WHERE name LIKE "Fred"
- SELECT name FROM mytable WHERE name LIKE "Fred%"

Prevent memory allocation error with Access 2000. V3.10(1012)

When viewing data in a linked ODBC table with Access 2000, if the table contained a string field whose length exceeded 256 bytes, the ODBC driver manager reported a memory allocation error. The problem was that Access 2000 was passing the ODBC driver a buffer that was too small to store the entire field, but the driver failed to return a “data truncated” warning. As a result, the SQL engine returned a large value (2GB) to the driver manager, which tried to allocate a buffer of this size. Now the driver returns a “data truncated” warning in this situation, which prevents the memory allocation error.

Function sequence error occurs when using multiple DAO recordsets with Access 2000. V3.10(1012)

Microsoft Access supports the creation of Visual Basic modules which can use the ODBC driver by way of Microsoft’s DAO (Data Access Objects) API. However, when using multiple recordsets, in some cases record retrieval operations fail with an “ODBC—call failed” error 3147 (although the same code works with with Access 97). The problem only occurs if the ODBC driver indicates that it supports transactions, so as a workaround the ODBC driver’s transaction support is now disabled in standalone mode.
### Multi-User Mode Fixes

**Error 36 occurs when reading data with c-tree application and ODBC driver at same time. V3.11(0827)**

A customer reported occurrences of error 36 when using both the ODBC driver and a c-tree V4 application to read data at the same time. The problem is that the ODBC driver uses “I/O locking” recommended by Microsoft to prevent caching of data by the network redirector, but the c-tree V4 application is not doing this. So, by locking data records in this manner, the ODBC driver prevents other c-tree applications from reading records. FairCom disabled the I/O locking, which is probably not needed anyway as the c-tree V4 ODBC driver only allows updates when opening the files in exclusive mode.

**Properly override alignment in multi-user mode. V3.11(0430)**

A customer reported a 445 error when using the current ODBC driver in multi-user mode. The problem was that the fields in the record were byte-aligned, but the alignment in the file’s header was set to 8. Forcing one-byte alignment using the ODBC driver’s Alignment setting did not resolve the problem. Using the ctalgn utility did solve the problem. The ODBC driver’s Alignment setting did not resolve the problem because in multi-user mode, the driver failed to indicate to c-tree that the alignment should be overridden. This is now done, resolving this problem.

**Prevent read/write errors due to I/O locking. V3.11(0316)**

Read or write errors may occur when using the c-tree ODBC Driver in a multi-user non-server networked environment in which some processes are using I/O locking (Windows 95 systems with old network redirectors) and others are not. If a process that is not using I/O locking tries to read or write a record locked by a process that is using I/O locking, the read/write will fail. The fix is as follows: if not using I/O locking, if a read or write fails due to the error ERROR_LOCK_VIOLATION, try to get a lock, and if the lock was successful, retry the read/write operation. This change was applied to the c-tree Plus code and was integrated into this build of the driver.

**Correct dynamic redirector detection. V3.10(1211)**

An additional change was applied to the function that detects the version of vredir.vxd, to prevent reading past the end of the work buffer. This prevents a potential crash when connecting to the driver on a Windows 95 system.

### Multithreading Fixes

**Crash when threads logon/logoff while other threads are using the driver. V3.11(0821)**

FairCom applied changes to the SQL engine to resolve a crash encountered by a customer when running his multi-threaded ODBC application.
Data Type Fixes

**Driver does not return full field contents for large CT_ARRAY (blob) fields. V3.11(0330)**

Prior to this build, the ODBC driver did not return the full contents of CT_ARRAY fields if the field data was over a certain length. The problem was that OT_GETIT() was limiting the number of bytes returned for CT_ARRAY fields to 16KB. Removing this restriction resolved the problem.

A symptom of this problem was that if a CT_ARRAY field was used to store OLE objects in a linked ODBC table in Microsoft Access, in some cases for large objects, the object could not be retrieved after it was stored.

**Prevent SCHEMAFieldLength warning on Windows 9x systems. V3.11(0316)**

When opening tables, the driver sometimes displayed a “SCHEMAFieldLength” warning message. This message only occurred on Windows 9x (not NT/2000 systems). The problem was due to the driver determining field alignment using an absolute memory address rather than an offset into a buffer. Because Windows 9x systems do not guarantee 8-byte alignment of allocated memory, using an absolute memory address to compute field alignment may give inconsistent results. The solution is to use an offset into the buffer instead of the memory address when computing field alignment. Applying this change resolved the problem.

SQL Engine Fixes

**“General ODBC Error” or incorrect results when sorting on non-indexed field. V3.10(1211)**

The SQL engine’s swap file logic potentially read past the end of a buffer, returning an error or incorrect data. The swap file logic is involved in queries that sort on a non-indexed field. This problem has been resolved.
12. New Function Descriptions

The following new functions have been added and are described in this chapter:

- CreateDataFileXtd8
- CreateIFileXtd8
- CreateIndexFileXtd8
- ctGETHGH
- ctMBprefix
- ctSETHGH
- ctThrdQueueReadDirect
- ctThrdQueueWriteDirect
- ctu16Tou8
- ctu8Tou16
- DropIndex
- GetServerInfoXtd
- GetXtdKeySegmentDef
- PermIIndex8
- PutIFileXtd8
- PutXtdKeySegmentDef
- RebuildIFileXtd8
- SetFileSegments
- TempIIndexXtd8
- TestHugeFile
- TransformXtdSegment
**CreateDataFileXtd8**

Extended 8 byte data file creation.

**SHORT NAME**  
CREDATX8

**TYPE**  
Low level 8 byte data file function

**DECLARATION**

```c
COUNT CreateDataFileXtd8(COUNT datno, pTEXT filnam,  
UCOUNT datlen, UCOUNT xtdsiz, COUNT filmod,  
LONG permmask, pTEXT groupid, pTEXT fileword,  
pXCREblk pxcreblk)
```

**DESCRIPTION**

`CreateDataFileXtd8` is a variation of `CreateDataFile` that permits the use of huge file support. This section expands on the description of `CreateDataFile` and `CreateDataFileXtd`.

`pxcreblk` points to a `XCREblk` structure, the extended creation block. For more information, review the "Huge File Support" chapter in this guide.

**RETURN**

`CreateDataFileXtd8` returns similar error codes to those for `CreateDataFile` and `CreateDataFileXtd`. See Appendix A of the c-tree Plus Programmer's Reference Guide for a complete listing of valid c-tree Plus error values.

**SEE ALSO**

`InitCTree`, `CreateIndexFile`, `CreateDataFile` and `CreateDataFileXtd`. 
CreateFileXtd8

Extended 8 byte Incremental ISAM creation.

SHORT NAME
CREIFILX8

TYPE
Extended 8 byte ISAM function

DECLARATION
COUNT CreateFileXtd8(pIFIL ifilptr, pTEXT dataextn,
pTEXT indxextn, LONG permmask, pTEXT groupid,
pTEXT fileword, pXCREblk pxcreblk)

DESCRIPTION
CreateFileXtd8 is a variation of CreateFile that permits the use of huge file support. This section expands on the description of CreateFile and CreateFileXtd.

pxcreblk points to an array of XCREblk structures, the extended creation block, one for each physical file specified in the IFIL structure, ifilptr. For more information, review the "Huge File Support" chapter in this guide.

RETURN
CreateFileXtd8 returns similar error codes to those for CreateFile and CreateFileXtd. See Appendix A of the c-tree Plus Programmer’s Reference Guide for a complete listing of valid c-tree Plus error values.

SEE ALSO
InitISAM, CreateFile, CreateFileXtd, OpenIFile, and CloseISAM.
**CreateIndexFileXtd8**

Extended 8 byte index file creation.

**SHORT NAME** CREIDX8

**TYPE** Low level data file function

**DECLARATION**

```c
COUNT CreateIndexFileXtd8(COUNT keyno, pTEXT filnam,
COUNT keylen, COUNT keytyp, COUNT keydup,
COUNT nomemb, UCOUNT xtdsiz, COUNT filmod,
LONG permask, pTEXT groupid, pTEXT fileword,
pXCREblk pxcreblk)
```

**DESCRIPTION**

CreateIndexFileXtd8 is a variation of CreateIndexFile that permits the use of huge file support. This section expands on the description of CreateIndexFile and CreateIndexFileXtd.

pxcreblk points to an XCREblk structure, the extended creation block. For more information, review the "Huge File Support" chapter in this guide.

**RETURN** CreateIndexFileXtd8 returns similar error codes to those for CreateIndexFile and CreateIndexFileXtd. See Appendix A of the c-tree Plus Programmer’s Reference Guide for a complete listing of valid c-tree Plus error values.

**SEE ALSO** InitCTree, CreateIndexFile, CreateDataFile and CreateIndexFileXtd.
ctGETHGH

Retrieve the high-order four bytes of an 8 byte record address.

SHORT NAME     ctGETHGH
TYPE            Low level function
DECLARATION     LONG ctGETHGH()
DESCRIPTION     Call ctGETHGH after a routine that returns or sets an output parameter to a record address to get the high-word portion of the address.

To minimize the effect on performance in client/Server environments, ctGETHGH does not make a separate call to the c-tree Server. Instead, the information needed by ctGETHGH is cached on the client side.

RETURN  ctGETHGH returns the high word portion of the address. See Appendix A of the c-tree Plus Programmer’s Reference Guide for a complete listing of valid c-tree Plus error values.

EXAMPLE    /* Assumes a key length of 14 (8 bytes for HUGE duplicate ** support), */
            /*
            LONG recadr_hw,recadr_lw;
            TEXT keybuf[14];

            /* return lower order 4 bytes of record address */
            recadr_lw = FirstKey(
                           9, /* index file number */
                           keybuf /* output key buffer */
                           );

            /* return higher order 4 bytes of record address */
            recadr_hw = ctGETHGH();
ctMBprefix

Stores a proper prefix to a Unicode filename string.

**SHORT NAME**
ctMBprefix

**TYPE**
Utility

**DECLARATION**
NINT ctMBprefix(pTEXT dp, NINT FnType)

**DESCRIPTION**
c-tree Plus requires that the Unicode file name have a special 8-byte prefix that informs c-tree Plus about special file name encoding. ctMBprefix stores a proper prefix at dp of type FnType.

dp is a pTEXT because the name may be encoded as a byte stream or a wide character array and ctMBprefix does not assume that dp is aligned when used with UTF16.

FnType may be ctFnTypeUTF8 or ctFnTypeUTF16. Both of these constants are defined in ctnb.h. ctFnPrefixSize holds the size of the prefix in bytes.

**RETURN**

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO_ERROR</td>
<td>Prefix stored.</td>
</tr>
<tr>
<td>446</td>
<td>BMOD_ERR</td>
<td>FnType parameter is bad</td>
</tr>
</tbody>
</table>

See Appendix A for a complete listing of valid c-tree Plus error values.

**EXAMPLE**

```c
/* In the following example, getUnicodeUTF8string is assumed ** to be a routine which will fill-in a UTF8 encoded string up ** to a max length. GetUnicodeUTF16string performs in the same ** manner except that it works with 16 bit wide characters. */

COUNT datno8, datno16;
WCHAR utf16name[256];
TEXT utf8name[512];

crMBprefix(utf8name, ctFnTypeUTF8);
getUnicodeUTF8string(utf8name + ctFnPrefixSize, 512 - ctFnPrefixSize);
datno8 = OPNRFIL(-1, utf8name, SHARED);

crMBprefix((pTEXT) utf16name, ctFnTypeUTF16);
getUnicodeUTF16string(utf16name + ctFnPrefixSize / 2, 256 - ctFnPrefixSize / 2);
datno16 = OPNRFIL(-1, utf16name, SHARED);
```
**ctSETHGH**

Set the high-order four bytes of an 8 byte record address.

**SHORT NAME**
ctSETHGH

**TYPE**
Low level function

**DECLARATION**

```c
NINT ctSETHGH(LONG highword)
```

**DESCRIPTION**

Call `ctSETHGH` before a routine requiring a record address as an input parameter to set the high word value for the function.

To minimize the effect on performance in client/Server environments, `ctSETHGH` does not make a separate call to the c-tree Server. Instead, the information supplied by `ctSETHGH` is cached on the client side.

**RETURN**

`ctSETHGH` always returns `NO_ERROR(0)`. See Appendix A of the *c-tree Plus Programmer’s Reference Guide* for a complete listing of valid *c-tree Plus* error values.

**EXAMPLE**

```c
/* Assumes a key length of 14 (8 bytes for HUGE duplicate ** support) */

LONG recadr_hw, recadr_lw;
pTEXT keyval;

keyval = "123456";

/* Remember, the high word counts the number of 4GB multiples ** contained in the composite 8 byte record address. Therefore, ** recadr_hw = 2 and recadr_lw = 512 means a record address of ** 8,589,935,104 */

recadr_hw = 2;
recadr_lw = 512;

/* set higher order 4 bytes of record address */

ctSETHGH(recadr_hw);

if (AddKey( /* add key value to index */
    9, /* index file number */
    keyval, /* pointer to key value */
    recadr_lw, /* lower order 4 bytes of record address */
    REGADD /* regular add mode */))
    printf("\nAddKey error = %d", uerr_cod);
```
**ctThrdQueueReadDirect**

Read a queue message pointer.

**SHORT NAME**
ctThrdQueueReadDirect

**TYPE**
Threading Function

**DECLARATION**

```c
NINT ctThrdQueueReadDirect(NINT qid, ppVOID msgadr,
                          pNINT pmsglen, LONG timeout)
```

**DESCRIPTION**

`ctThrdQueueReadDirect` places the address of the next message from queue `qid` into `msgadr`, and places the length of the message in `pmsglen`. `ctThrdQueueReadDirect` does not copy the message contents.

**NOTE:** Your application is responsible for managing the buffers used in the read and write calls. To avoid memory/buffer errors, FairCom recommends using either the original Queue read/write calls or the new direct calls, but not mixing calls from both pairs.

The return value is an error indicator.

**RETURN**

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO_ERROR</td>
<td>Successful convert.</td>
</tr>
<tr>
<td>90</td>
<td>NQUE_ERR</td>
<td>Invalid <code>qid</code>.</td>
</tr>
<tr>
<td>156</td>
<td>NTIM_ERR</td>
<td>Timeout error.</td>
</tr>
<tr>
<td>514</td>
<td>CQUE_ERR</td>
<td>Queue could not be closed.</td>
</tr>
</tbody>
</table>

See Appendix A for a complete listing of valid c-tree Plus error values.

**SEE ALSO**

`ctThrdQueueRead` and `ctThrdQueueWriteDirect`
**ctThrdQueueWriteDirect**

Write a queue message pointer.

**SHORT NAME**

ctThrdQueueWriteDirect

**TYPE**

Threading Function

**DECLARATION**

NINT ctDECL ctThrdQueueWriteDirect(NINT qid, pVOID message, NINT msglen)

**DESCRIPTION**

ctThrdQueueWriteDirect adds the address for queue entry *message* of length *msglen* to the queue *qid*. Unlike ctThrdQueueWrite, which copies the contents of the entry into a buffer allocated by the queue write call, ctThrdQueueWriteDirect. Uses pointers to buffers created by the application to manage messages.

**NOTE:** The application is responsible for managing the buffers used in the ctThrdQueueReadDirect and ctThrdQueueWriteDirect calls. To avoid memory/buffer errors, FairCom recommends using either the original Queue read/write calls or the new direct calls, but not mixing calls from both pairs.

The return value is an error indicator.

**RETURN**

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO_ERROR</td>
<td>Successful queue write.</td>
</tr>
<tr>
<td>90</td>
<td>NQUE_ERR</td>
<td>Invalid <em>qid</em>.</td>
</tr>
<tr>
<td>92</td>
<td>QMRT_ERR</td>
<td>Queue memory error.</td>
</tr>
<tr>
<td>514</td>
<td>CQUE_ERR</td>
<td>Queue could not be closed.</td>
</tr>
</tbody>
</table>

See Appendix A for a complete listing of valid c-tree Plus error values.

**SEE ALSO**

ctThrdQueueWrite and ctThrdQueueReadDirect
ctu16TOu8

Converts a UTF16 Unicode string to a UTF8 encoded string.

SHORT NAME  
ctu16TOu8

TYPE  
Utility

DECLARATION  
NINT ctu16TOu8(pWCHAR u16str, pTEXT u8str, VRLEN u8byt)

DESCRIPTION  
ctu16TOu8 converts a UTF16 Unicode string to a UTF8 encoded string. The input strings are assumed to be terminated by a NULL character. All output buffer sizes are specified in bytes.

RETURN

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO_ERROR</td>
<td>Successful convert.</td>
</tr>
<tr>
<td>153</td>
<td>VBSZ_ERR</td>
<td>The output buffer is too small.</td>
</tr>
<tr>
<td>446</td>
<td>BMOD_ERR</td>
<td>There is a problem with the input string.</td>
</tr>
</tbody>
</table>

See Appendix A for a complete listing of valid c-tree Plus error values.

SEE ALSO  
ctu8TOu16
ctu8TOu16

Converts an ASCII or UTF8 Unicode string to a UTF16 encoded string.

SHORT NAME  ctu8TOu16
TYPE        Utility
DECLARATION NINT ctu8TOu16 (pTEXT u8str, pWCHAR u16str, VRLEN u16byt)
DESCRIPTION ctu8TOu16 converts an ASCII or UTF8 Unicode string to a UTF16 encoded string. The input strings are assumed to be terminated by a NULL character. All output buffer sizes are specified in bytes.

RETURN

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO_ERROR</td>
<td>Successful convert.</td>
</tr>
<tr>
<td>153</td>
<td>VBSZ_ERR</td>
<td>The output buffer is too small.</td>
</tr>
<tr>
<td>446</td>
<td>BMOD_ERR</td>
<td>There is a problem with the input string.</td>
</tr>
</tbody>
</table>

See Appendix A for a complete listing of valid c-tree Plus error values.

SEE ALSO  ctu16TOu8
**DropIndex**

Permanent Incremental Index deletion.

<table>
<thead>
<tr>
<th>SHORT NAME</th>
<th>ctDROPIDX</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>ISAM Function</td>
</tr>
<tr>
<td>DECLARATION</td>
<td>\texttt{COUNT \ DropIndex(COUNT keyno)}</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td><strong>DropIndex</strong> permanently removes the index file referenced by \textit{keyno}, which must be associated with a host index, not a member index. All the indices contained in the index file are dropped and the IFIL resource in the data file is updated. If the \textit{keyno} index is transaction dependent, \texttt{TRANDEP}, <strong>DropIndex</strong> must be performed under transaction control and can be undone by an abort or save point restore, but they are marked so that no ISAM update will touch these indices. <strong>NOTE:</strong> The key numbers associated with the data file may have a gap until the files are closed and reopened. For example, if a data file has key #’s 3, 4, 5, 6, 7, and 8 associated, and key 5 is its own file with one member, then after <strong>DropIndex</strong>(5) completes the key numbers associated with the data file will be: 3, 4, 7, and 8. After the data file is closed and reopened (at the ISAM level), the key numbers will be consecutive again: 3, 4, 5, and 6.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RETURN</th>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>NO_ERROR</td>
<td>Successful drop of index file.</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>KMEM_ERR</td>
<td>Attempt to drop a member index.</td>
</tr>
<tr>
<td></td>
<td>48</td>
<td>FMOD_ERR</td>
<td>keyno is not an index file.</td>
</tr>
<tr>
<td></td>
<td>62</td>
<td>LERR_ERR</td>
<td>File must be opened EXCLUSIVE.</td>
</tr>
<tr>
<td></td>
<td>446</td>
<td>BMOD_ERR</td>
<td>Attempt to drop a temporary index.</td>
</tr>
</tbody>
</table>

**LIMITATIONS**  
**DropIndex** cannot be called for indices created by **TempIIndexXtd** or for member indices. Specify a permanent host index to be dropped.  
Both **PermIIndex** and **DropIndex** are only undoable if they are performed on **TRANDEP** files.  
If there are pending **DropIndex** calls, then no calls can be made to **UpdateConditionalIndex** for any of the indices associated with the underlying data file.

**SEE ALSO**  
**PermIIndex**
### GetServerInfoXtd

Listen for server information via TCP/IP, Extended Version.

**SHORT NAME**
GetServerInfoXtd

**TYPE**
Low-level Function

**DECLARATION**
```c
NINT GetServerInfoXtd(pTEXT buffer, NINT bufsiz, UCOUNT port, LONG sec)
```

**DESCRIPTION**
GetServerInfoXtd listens for a broadcasting c-tree Server on TCP/IP port `port` for `sec` seconds before timing out. The `port` value should match the TCP/IP port set in the Server `BROADCAST_PORT` keyword. A negative `sec` defaults to 90 seconds. If a broadcast is received, up to `bufsiz` bytes are stored at `buffer`.

**RETURN**
This routine returns `NO_ERROR` if executed properly, `-1` if the `buffer` was smaller than the string it received, or the value of the system-level `errno` if an error occurred. Check your compiler’s documentation for valid `errno` values.

**EXAMPLE**
```c
TEXT info_buf[128];
NINT retval;
pTEXT wptr;
UCOUNT port = myBroadcastPort;
LONG sec = myBroadcastTimeout;

crt_printf("Checking for Broadcast Servers .........
");
if (retval = GetServerInfoXtd(info_buf, (NINT) sizeof(info_buf)
    port, sec)) {
    crt_printf("Error getting available servers [%ld] \n",
        (LONG)retval);
}

if ((crt_strlen(info_buf) < 1) && !retval) {
    printf("No Broadcasting Servers Available \n");
} else {
    if ((wptr = strchr(info_buf,'|'))
        *wptr = '@';
    if ((wptr = strchr(info_buf,'|'))
        *wptr = '\0';
    crt_printf("We will attempt a connection to [%s] on port
        [%s].\n", info_buf, ++wptr);
    *svn = info_buf;
}
```

**SEE ALSO**
GetServerInfo
**GetXtdKeySegmentDef**

Retrieves the requested extended key segment definition.

**SHORT NAME**
GETKSEGDEF

**TYPE**
ISAM Data Definition

**DECLARATION**
NINT GetXtdKeySegmentDef(NINT filno, NINT segno, pctlKSEGDEF pkdef)

**DESCRIPTION**
GetXtdKeySegmentDef retrieves (i.e., fills in the elements of a ctKSEGDEF structure for) the requested extended key segment definition to pkdef, except that the kseg_type member of the pkdef structure should be set on input to the type of segment to be retrieved.

For example, to retrieve an ICU Unicode definition, set the kseg_type member to ctKSEG_TYPE_UNICODE before calling GetXtdKeySegmentDef. If GetXtdKeySegmentDef is called with ctKSEGhandle for the filno parameter and the handle value is passed in via the segno parameter (as shown in the third row in the table below), then the kseg_type member of the structure is ignored on input since the handle uniquely identifies the particular definition. On output, the kseg_type member will be set to the type of segment.

One of the main reasons to call GetXtdKeySegmentDef is to be able to examine the actual locale being used for the ICU collation routines.

The filno and segno values determine which definition is returned, as follows:

<table>
<thead>
<tr>
<th>filno</th>
<th>segno</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctKSEGserver</td>
<td>ignored</td>
<td>Retrieve server default definition</td>
</tr>
<tr>
<td>ctKSEGapplic</td>
<td>ignored</td>
<td>Retrieve application default definition</td>
</tr>
<tr>
<td>ctKSEGhandle</td>
<td>handle</td>
<td>Retrieve definition associated with handle</td>
</tr>
<tr>
<td>datno</td>
<td>ignored</td>
<td>Retrieve data file level definition</td>
</tr>
<tr>
<td>keyno</td>
<td>ctKSEGindex</td>
<td>Retrieve index file level definition</td>
</tr>
<tr>
<td>keyno</td>
<td>0, 1, 2, ...</td>
<td>Retrieve particular segment definition</td>
</tr>
</tbody>
</table>
RETURN

If successful, `GetXtdKeySegmentDef` returns the handle associated with the definition. `GetXtdKeySegmentDef` returns a negative value upon error, where the absolute value of the return value is the error code. The most common errors are shown below. See Appendix A of the *c-tree Plus Programmer’s Reference Guide* for a complete listing of valid *c-tree Plus* error values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>437</td>
<td>DADR_ERR</td>
<td>NULL <code>pkdef</code> argument.</td>
</tr>
<tr>
<td>694</td>
<td>NUNC_ERR</td>
<td>Executable does not support ICU Unicode, but a UNCSEG modifier has been encountered.</td>
</tr>
<tr>
<td>701</td>
<td>CSEG_ERR</td>
<td>Could not process the <code>kseg_comp</code> options. This could occur if more than one of a set of mutually exclusive options are combined.</td>
</tr>
<tr>
<td>702</td>
<td>ASEG_ERR</td>
<td>An error occurred when attempting to process one of the special attribute options.</td>
</tr>
<tr>
<td>703</td>
<td>HSEG_ERR</td>
<td>Invalid key segment handle. The <code>segno</code> parameter should be set to a valid extended key segment handle.</td>
</tr>
<tr>
<td>706</td>
<td>NSEG_ERR</td>
<td>Zero bytes of binary sort key were generated. Possibly an all NULL source.</td>
</tr>
<tr>
<td>707</td>
<td>USEG_ERR</td>
<td>There is no extended key segment definition to use.</td>
</tr>
<tr>
<td>708</td>
<td>MBSP_ERR</td>
<td>Multibyte/Unicode file names are not supported.</td>
</tr>
<tr>
<td>709</td>
<td>MBNM_ERR</td>
<td>A badly formed multibyte/Unicode file name has been encountered.</td>
</tr>
<tr>
<td>710</td>
<td>MBFM_ERR</td>
<td>A multibyte/Unicode variant is not supported (e.g., UTF32).</td>
</tr>
</tbody>
</table>

EXAMPLE

See the API example in the Unicode chapter of this guide.

SEE ALSO

`PutXtdKeySegmentDef` and `TransformXtdSegment`. 
**PermIIndex8**

Permanent 8 byte Incremental Index creation.

<table>
<thead>
<tr>
<th>SHORT NAME</th>
<th>PRMIIDX8</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Extended 8 byte ISAM function</td>
</tr>
<tr>
<td>DECLARATION</td>
<td><code>COUNT PermIIndex8(pIFIL ifilptr, pXCREblk pxcreblk)</code></td>
</tr>
</tbody>
</table>
| DESCRIPTION  | *PermIIndex8* is a variation of *PermIIndex* that permits the use of huge file support. This section expands on the description of *PermIIndex*.*

`pxcreblk` points to an array of `XCREblk` structures, the extended creation block, one for each physical file in `ifilptr`. For more information, review the "Huge File Support" chapter in this guide.

<table>
<thead>
<tr>
<th>RETURN</th>
<th><em>PermIIndex8</em> returns error codes similar to those for <em>PermIIndex</em>. See Appendix A of the <em>c-tree Plus Programmer’s Reference Guide</em> for a complete listing of valid <em>c-tree Plus</em> error values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIMITATIONS</td>
<td><em>PermIIndex8</em> and <em>TempIIndexXtd</em> support <em>ctTRANDEP</em> creates. Without <em>ctTRANDEP</em> creates, these routines cannot be called within a transaction. With <em>ctTRANDEP</em> creates, they must be called within a transaction.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td><em>TempIIndexXtd, RebuildIIndex, PermIIndex, and UpdateConditionalIndex.</em></td>
</tr>
</tbody>
</table>
PutIFILXtd8

Put an IFIL structure into a data file resource record. Extended 8 byte version.

SHORT NAME  PUTIFILX8

TYPE  Extended 8 byte ISAM Function

DECLARATION  COUNT PutIFILXtd8(pIFIL ifilptr, pTEXT dataextn,
                             pTEXT indxextn, pTEXT fileword, pXCREblk pxcreblk)

DESCRIPTION  PutIFILXtd8 functions exactly as PutIFILXtd. It accepts a pointer to an extended
              file creation block, pxcreblk, but does not currently use that information. It is
              intended as a placeholder for a future 8 byte version of PutIFILXtd8.

RETURN

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO_ERROR</td>
<td>IFIL resource stored successfully.</td>
</tr>
</tbody>
</table>

See Appendix A for a complete listing of valid c-tree Plus error values.

EXAMPLE

```c
    IFIL vc_dat = {
        // ... IFIL structure to be placed in resource record */
    }
    COUNT ret=0; /* function return work variable */
    pTEXT fileword;

    getfileword(&fileword);
    if (ret = PutIFILXtd8(&vc_dat, NULL, NULL, fileword, NULL))
        printf("\nError during PUTIFILX8, error = %d",ret);
    else
        printf("\nSuccessful PUTIFILX8! ");
```

SEE ALSO  PutIFIL and PutIFILXtd
PutXtdKeySegmentDef

 Defines an extended key segment for a Server, an application, a data file, an index file, or a particular index segment.

**SHORT NAME**
PUTKSEGDEF

**TYPE**
ISAM Data Definition

**DECLARATION**

```c
NINT PutXtdKeySegmentDef(NINT filno, NINT segno,
                           pctKSEGDEF pkdef)
```

**DESCRIPTION**

PutXtdKeySegmentDef defines an extended key segment for a Server, an application, a data file, an index file, or a particular index segment. The `pkdef` parameter points to a definition to be created in a call to PutXtdKeySegmentDef. The `filno` and `segno` parameters determine where the definition is to be stored, as follows:

<table>
<thead>
<tr>
<th><code>filno</code></th>
<th><code>segno</code></th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctKSEGserver</td>
<td>ignored</td>
<td>Create server default definition</td>
</tr>
<tr>
<td>ctKSEGapplic</td>
<td>ignored</td>
<td>Create application default definition</td>
</tr>
<tr>
<td>datno</td>
<td>ignored</td>
<td>Create data file level definition</td>
</tr>
<tr>
<td>keyno</td>
<td>ctKSEGindex</td>
<td>Create index file level definition</td>
</tr>
<tr>
<td>keyno</td>
<td>0, 1, 2, ...</td>
<td>Create specific segment definition</td>
</tr>
</tbody>
</table>

**RETURN**

PutXtdKeySegmentDef returns a handle for the definition if successful. PutXtdKeySegmentDef returns a negative value upon error, where the absolute value of the return value is the error code. The most common errors are shown below. See Appendix A of the c-tree Plus Programmer’s Reference Guide for a complete listing of valid c-tree Plus error values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>LERR_ERR</td>
<td>PutXtdKeySegmentDef called for a data or index file requires the file to be opened exclusively. A just created file is in exclusive mode, regardless of the specified file mode, until it is closed and re-opened.</td>
</tr>
<tr>
<td>437</td>
<td>DADR_ERR</td>
<td>NULL <code>pkdef</code> argument.</td>
</tr>
<tr>
<td>445</td>
<td>SDAT_ERR</td>
<td>No source data to create key segment.</td>
</tr>
<tr>
<td>446</td>
<td>BMOD_ERR</td>
<td>Improper <code>filno</code> or <code>segno</code> values if the handle references an extended key segment definition not supported by the executable.</td>
</tr>
<tr>
<td>589</td>
<td>LADM_ERR</td>
<td>Only an ADMIN group member may set a Server default (i.e., ctKSEGserver).</td>
</tr>
</tbody>
</table>
# New Function Descriptions

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>694</td>
<td>NUNC_ERR</td>
<td>Executable does not support ICU Unicode, but a UNCSEG modifier has been encountered.</td>
</tr>
<tr>
<td>700</td>
<td>OSEG_ERR</td>
<td>Could not process key segment definition.</td>
</tr>
<tr>
<td>701</td>
<td>CSEG_ERR</td>
<td>Could not process the \texttt{kseg_comp} options. This could occur if more than one of a set of mutually exclusive options are combined.</td>
</tr>
<tr>
<td>702</td>
<td>ASEG_ERR</td>
<td>An error occurred when attempting to process one of the special attribute options.</td>
</tr>
<tr>
<td>703</td>
<td>HSEG_ERR</td>
<td>Invalid key segment handle in a call to \texttt{TransformXtdSegment} or in a call to \texttt{GetXtdKeySegmentDef} when the \texttt{ct_KSEG_handle} option is used and the \texttt{segno} parameter should be set to a valid extended key segment handle.</td>
</tr>
<tr>
<td>704</td>
<td>SSEG_ERR</td>
<td>No source type provided when \texttt{kseg_styp} has been set to \texttt{ct_KSEG_STYP_PROVIDED}. If this error occurs, it is likely to occur during the first use (say with an \texttt{AddRecord} or \texttt{AddVRecord} or \texttt{OpenIFile}) of the extended key segment.</td>
</tr>
<tr>
<td>705</td>
<td>DSEG_ERR</td>
<td>An extended key segment definition already exists at the level implied by the \texttt{PutXtdKeySegmentDef} call.</td>
</tr>
<tr>
<td>706</td>
<td>NSEG_ERR</td>
<td>Zero bytes of binary sort key were generated. Possibly an all NULL source.</td>
</tr>
<tr>
<td>707</td>
<td>USEG_ERR</td>
<td>There is no extended key segment definition to use.</td>
</tr>
<tr>
<td>708</td>
<td>MBSP_ERR</td>
<td>Multibyte/Unicode file names are not supported.</td>
</tr>
<tr>
<td>709</td>
<td>MBNM_ERR</td>
<td>A badly formed multibyte/Unicode file name has been encountered.</td>
</tr>
<tr>
<td>710</td>
<td>MBFM_ERR</td>
<td>A multibyte/Unicode variant is not supported (e.g., UTF32)</td>
</tr>
</tbody>
</table>

**EXAMPLE**  
See the API example in the Unicode chapter of this guide.

**SEE ALSO**  
\texttt{GetXtdKeySegmentDef} and \texttt{TransformXtdSegment}.
RebuildIFileXtd8

Extended 8 byte Incremental ISAM rebuild.

SHORT NAME  RBLIFILX8
TYPE           Extended 8 byte ISAM function
DECLARATION    COUNT RebuildIFileXtd8(pIFIL ifilptr, pTEXT dataextn,
                                 pTEXT indexextn, LONG permmask, pTEXT groupid,
                                 pTEXT fileword, pXCREblk pxcreblk)
DESCRIPTION    RebuildIFileXtd8 is a variation of RebuildIFile that permits the use of huge file
               support. This section expands on the description of RebuildIFile and
               RebuildIFileXtd.
               pxcreblk points to an array of XCREblk structures, the extended creation block, one
               for each physical file in ifilptr. For more information, review the "Huge File Support"
               chapter in this guide.
RETURN         RebuildIFileXtd8 returns error codes similar to those for RebuildIFile and
               RebuildIFileXtd. See Appendix A of the c-tree Plus Programmer’s Reference
               Guide for a complete listing of valid c-tree Plus error values.
SEE ALSO       InitISAM, RebuildIFile, CloseISAM, and RebuildIFileXtd.
**SetFileSegments**

Segmented file support configuration function.

**SHORT NAME**
ctSETSEG

**TYPE**
Low level function

**DECLARATION**
COUNT SetFileSegments(COUNT filno, NINT aseg, NINT tseg, pSEGMDEF pseg)

**DESCRIPTION**

SetFileSegments configures the segments for file filno. aseg specifies the number of active segments, i.e., the segments created immediately. aseg must be at least one and less than or equal to tseg. tseg specifies the total number of segments pointed to by pseg. pseg points to an array of SEGMDEF structures.

If the first segment definition pointed to by pseg has a sgname pointing to the empty string, i.e., *sgname == '\0', not sgname == NULL, the sgsizemember of the structure becomes the host segment size limit. Only the last segment can have a size limit of zero, which is interpreted as no limit.

Additional segments automatically become active as needed, up to the maximum set in tseg. The segments are used in the order defined by the array of SEGMDEF structures pointed to by pseg.

The file referenced by filno must be opened in ctEXCLUSIVE mode the first time SetFileSegments is called. Note that a file which has been created and not yet closed is always in ctEXCLUSIVE mode, regardless of the file mode specified as part of the create call. After the segment definitions have been established by the first call to SetFileSegments, it is possible to call SetFileSegments to modify the segment definitions even while the file is being updated. However, it is not possible to change a segment size so that the new size is smaller than the actual physical size of the segment, nor can SetFileSegments rename a segment that is in use. A segment is in use if data beyond the segment header information has been written to the segment. An active segment is not in use just because it is on disk; data must have been written to it. Therefore, a call to SetFileSegments can, in real time, change where segments will reside (provided the segment is not already in use) and/or how large they are (provided the new size is not smaller than the current physical size nor is the segmented already completely full).
### RETURN

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO_ERROR</td>
<td>Successful segment configuration.</td>
</tr>
<tr>
<td>62</td>
<td>LERR_ERR</td>
<td>First call requires exclusive file lock.</td>
</tr>
<tr>
<td>70</td>
<td>TEXS_ERR</td>
<td>Call cannot be made within a transaction unless the file has just been created with <code>ctTRANDEF</code> or <code>ctRSTRDEL</code> (i.e., the creation has not been committed) and this is the first such call for this file.</td>
</tr>
<tr>
<td>446</td>
<td>BMOD_ERR</td>
<td>Illegal value for <code>aseg</code>.</td>
</tr>
<tr>
<td>448</td>
<td>DEFP_ERR</td>
<td>Denied permission to change file definition.</td>
</tr>
<tr>
<td>454</td>
<td>NSUP_ERR</td>
<td>File requires extended header to support segments.</td>
</tr>
</tbody>
</table>


### EXAMPLE

```c
SEGDEF segdef[2] = {
    {"d:\data\dataseg.2",1024}, /* 1024MB = 1GB size limit */
    {"e:\data\dataseg.3",0}    /* no limit on segment size */
};

IFIL fil = {...}
/* create files */
if (CreateIFileXtd8( &fil, NULL, NULL, 0L, NULL, NULL,
    &creblk)) /* pointer to array of XCREblks */
    printf("Could not create %s (%d,%d)
", fil.pfilnam,
    isam_err, isam_fil);

/* specify definitions for the two other segments */
if (SetFileSegments(
    fil.tfilno, /* data file number */
    1, /* one active segment (the host segment) */
    2, /* two segment definitions to be passed */
    segdef)) /* pointer to the segment definitions */
    printf("Could not set file segments (%d,%d)
", isam_err,
    isam_fil);
```

### LIMITATIONS

The `fxtsiz` member of the XCREblk structure cannot be set higher than the size of the first (host) segment during a file create. This results in `SEG_ERR(674)` signifying the need for more segments, which do not exist yet because `SetFileSegments` has not yet been called.

The file referenced by `filno` must be opened in `ctEXCLUSIVE` mode the first time `SetFileSegments` is called.
TempIIndexXtd8

Extended 8 byte temporary Incremental Index creation.

SHORT NAME    TMPIIDXX8
TYPE           Extended 8 byte ISAM function
DECLARATION    COUNT TempIIndexXtd8(pIFIL ifilptr, LONG permmask,
                                 pTEXT groupid, pTEXT fileword, pXCREblk pxcreblk)
DESCRIPTION    TempIIndexXtd8 is a variation of TempIIndexXtd that permits the use of huge file support. This section expands on the description of TempIIndexXtd.

pxcreblk points to an array of XCREblk structures, the extended creation block, one for each physical file in ifilptr. For more information, review the "Huge File Support" chapter in this guide.

RETURN        TempIIndexXtd8 returns error codes similar to those for TempIIndexXtd. See Appendix A of the c-tree Plus Programmer’s Reference Guide for a complete listing of valid c-tree Plus error values.

LIMITATIONS   PermIIndex8 and TempIIndexXtd8 support ctTRANDEP creates. Without ctTRANDEP creates, these routines cannot be called within a transaction. With ctTRANDEP creates, they must be called within a transaction.

SEE ALSO      PermIIndex, RebuildIIndex, and TempIIndexXtd.
**TestHugeFile**

Test 8 byte file status.

**SHORT NAME**
TESTHUGE

**TYPE**
Low level function

**DECLARATION**
COUNT TestHugeFile(COUNT filno)

**DESCRIPTION**
TestHugeFile takes the file number of an opened file, *filno*, as its input argument to determine if the volume containing the file supports huge files.

**NOTE:** This function does NOT determine how much space is available, only if the OS logically supports such file sizes on the volume containing the open file referenced by *filno*.

**RETURN**
TestHugeFile returns the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO_ERROR</td>
<td>Files &gt; 4GB supported</td>
</tr>
<tr>
<td>-647</td>
<td>E2GB_COD</td>
<td>Files &gt; 2GB NOT supported</td>
</tr>
<tr>
<td>-648</td>
<td>E4GB_COD</td>
<td>Files &gt; 4GB NOT supported</td>
</tr>
<tr>
<td>26</td>
<td>FACS_ERR</td>
<td>if <em>filno</em> is not in use</td>
</tr>
</tbody>
</table>

**TransformXtdSegment**

Creates a binary sort key (segment) using an extended key segment definition.

**SHORT NAME**  
XFMKSEGDEF

**TYPE**  
ISAM Data Definition

**DECLARATION**  
NINT TransformXtdSegment(NINT seghnd, pVOID src, NINT srclen, NINT srctyp, pVOID dest, NINT destlen)

**DESCRIPTION**  
Most applications will not have a reason to call **TransformXtdSegment** unless the application needs to create a Unicode binary sort key outside of the normal ISAM processing.

Creates a binary sort key (segment) using an extended key segment definition.

<table>
<thead>
<tr>
<th>seghnd</th>
<th>Handle returned by <strong>PutXtdKeySegmentDef</strong> or <strong>GetXtdKeySegmentDef</strong>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>Pointer to data used to construct segment.</td>
</tr>
<tr>
<td>srclen</td>
<td>Size in bytes of the region pointed by <em>src</em>. However, <em>srclen</em> is ignored unless <em>kseg_ssz</em> was set to ctKSEG_SSSZ_PROVIDED.</td>
</tr>
<tr>
<td>srctyp</td>
<td><em>srctyp</em> should be set to one of the c-tree Plus field types (e.g., CT_STRING or CT_UNICODE). However, <em>srctyp</em> is ignored unless <em>kseg_styp</em> was set to ctKSEG_STYP_PROVIDED.</td>
</tr>
<tr>
<td>dest</td>
<td>Pointer to region in which binary sort key is constructed.</td>
</tr>
<tr>
<td>destlen</td>
<td>Size in bytes of the region pointed to by <em>dest</em>.</td>
</tr>
</tbody>
</table>

**RETURN**  
If successful, it returns the number of bytes used for the binary sort key. Returns a negative value upon error, where the absolute value of the return value is the error code. The most common errors are shown below. See Appendix A of the c-tree Plus Programmer’s Reference Guide for a complete listing of valid c-tree Plus error values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>445</td>
<td>SDAT_ERR</td>
<td>No source data to create key segment.</td>
</tr>
<tr>
<td>446</td>
<td>BMOD_ERR</td>
<td>The handle references an extended key segment definition not supported by the executable.</td>
</tr>
<tr>
<td>694</td>
<td>NUNC_ERR</td>
<td>Executable does not support ICU Unicode, but a UNCSEG modifier has been encountered.</td>
</tr>
<tr>
<td>700</td>
<td>OSEG_ERR</td>
<td>Could not process key segment definition.</td>
</tr>
<tr>
<td>701</td>
<td>CSEG_ERR</td>
<td>Could not process the <em>kseg_comp</em> options. This could occur if more than one of a set of mutually exclusive options are combined.</td>
</tr>
</tbody>
</table>
## New Function Descriptions

**Value** | **Symbolic Constant** | **Explanation**
---|---|---
702 | ASEG_ERR | An error occurred when attempting to process one of the special attribute options.
703 | HSEG_ERR | Invalid key segment handle.
704 | SSEG_ERR | No source type provided when kseg_styp has been set to ct.KSEG_STYP_PROVIDED. If this error occurs, it is likely to occur during the first use (say with an AddRecord or AddVRecord or OpenIFile) of the extended key segment.
706 | NSEG_ERR | Zero bytes of binary sort key were generated. Possibly an all NULL source.
707 | USEG_ERR | There is no extended key segment definition to use.
708 | MBSP_ERR | Multibyte/Unicode file names are not supported.
709 | MBNM_ERR | A badly formed multibyte/Unicode file name has been encountered.
710 | MBFM_ERR | A multibyte/Unicode variant is not supported (e.g., UTF32).

**EXAMPLE**

See the API example in the Unicode chapter of this guide.

**SEE ALSO**

GetXtdKeySegmentDef and PutXtdKeySegmentDef.
13. Fixes

The following issues represent critical issues and other fixes corrected in recent builds. Entries are marked with the build date in sequential order beginning with V7.10 (1204).

13.1 Critical Issues

Critical issues represent serious problems resulting in a crash or data corruption. Though most are unlikely to occur, they have serious consequences when they do.

**Dynamic Dump Restore BNOD_ERR(69) – (1018)**

An issue with the c-tree Server restore process caused selected indices to not be addressed properly during the restore procedure. The pivot point of the problem was a subtle issue related to when transactions completed and when the dynamic dump process issued its final checkpoint. Specifically, an index that had just experienced activity under transaction control and had been committed was being detected as "in-order", when, in fact, it required additional consideration by the ctrdmp recovery routine. Data files were not affected and a rebuild corrected the issue.

**Automatic Recovery Index Errors (0808)**

An issue that could result in index errors, such as BNOD_ERR(69), during or after automatic recovery or dynamic dump restore has been fixed. Automatic recovery of indices with multiple members infrequently resulted in damage to the headers. Data files were not affected and a rebuild corrected the issue.

**Incorrect End-of-File Value Restored When 4GB Limit Exceeded (0525)**

When an attempt was made to exceed the 4 GB file limit for a non-huge file, the file extension routine would correctly detect that the limit was exceeded and return a FULL_ERR(39). Similarly, on a disk full check, the FULL_ERR(39) would be returned if the volume was full. However, the logical end-of-file value in the header was not being restored properly. After the error occurred, new records could be added to the file, but they would overwrite the beginning of the file. Note that the information that was overwritten would not necessarily start at the absolute beginning of the file, depending on how the new space request wrapped around the 4 GB limit. This issue has been resolved so that the header value is updated correctly.

**Dynamic Dump Roll Forward Error Resolved (0424)**

An issue has been corrected in the dynamic dump restore utility, ctrdmp, that could cause a RFCK_ERR(510) while attempting to roll forward from a dynamic dump restore. The issue involved the internal checkpoint handling. There was a chance that a roll forward following a dynamic dump restore could start from the wrong checkpoint. This was not an issue with a roll forward from a standard backup.
This is considered a critical problem. All users who utilize the ctrdmp and cfldmp utilities to roll forward from a dynamic dump restore should update immediately to.

**Huge Files Not Created on 64 Bit Platforms (0420)**

An error that prevented files from being created as huge files was corrected. This was only an issue with Servers hosted on native 64 bit platforms.

**Serious Data Cache Issue Resolved (0201)**

An issue was discovered related to the c-tree Server internal cache management. All previous V7.10 users MUST update their use of the Server to prevent a potentially serious data corruption issue. This did not affect FairCom Server V6.xx.

### 13.2 Serious Issues

**DoBatch: 4GB Boundary Issue Resolved (0808)**

An obscure problem was resolved related to the use of DoBatch: If, by chance, a record fell on a 4GB boundary, the internal logic within c-tree Plus would fail to return the entire batch. DoBatch now returns the entire batch correctly.

**Corrected PREIMAGE_DUMP catend (0717)**

A dynamic dump restore, ctrdmp, could terminate with an L64 error on PREIMG files dumped with the PREIMAGE_DUMP keyword. A status value was altered to correct this issue.

**Threading Semaphore (0717/0816)**

An internal semaphore issue, which caused the Server to fail under QNX RTOS, has been resolved. This change was applied to all Unix platforms in the 0816 release.

**DoBatch Issue With 8 byte Offsets (0717)**

DoBatch calls using BAT_NXT returning record offsets were only returning 4 byte offsets. The routines were corrected to properly return 4 byte or 8 byte offsets, as appropriate.

**CompactlFile Errors DLOP_ERR(512) and RRED_ERR(407) (0615)**

A customer encountered DLOP_ERR(512) and RRED_ERR(407) with a bound server call to the CompactlFile. FairCom made internal code changes to initialize required variables and set appropriate input parameters to match the HUGE status of the file to be compacted.

**Windows Shared Memory Logon Issues Resolved (0509)**

A subtle problem related to logging on a c-tree Server has been resolved. When rapid logon/logoff sequences were pursuing the c-tree Server, the internal shared memory
pipe response would return a “busy” error, thus denying the client’s the ability to logon. By adding internal retry logic, this issue is now resolved. Any users who experienced occasional logon failures during high volume of logon/logoff activity should appreciate this fix.

**BAT RET KEY and uTFRMKEY (0509)**

DoBatch called with BAT RET KEY was returning keys in the byte-order native to the client. This has been adjusted to work as documented, returning keys in native index format (i.e., HIGH_LOW).

To restore the old behavior (keys returned in client numeric format) in Standalone models, add #define ctBEHAV_BATUTFRM at compile time.

To restore the old behavior (keys returned in client numeric format) with the Server, add COMPATIBILITY BATCH.UTFFRMKEY to the Server configuration.

**RVHD_ERR(123) Condition Resolved (0509)**

A internal problem which generated a RVHD_ERR(123) error while adding a record has been resolved. The problem was qualified to transaction-controlled files, and would occur after an aborted transaction when the new record tried to reuse the aborted space.

**Solaris Bound Server Hang Resolved (0509)**

A rare and unusual occurrence could hang a thread on a Bound Server under Solaris. Checking for a valid parameter value resolved the issue.

**HP-UX 11 Update (0424)**

A problem related to incorrect compile switches on HP-UX 11 has been resolved. Customers who are developing applications for the HP-UX 11 platform should update immediately.

**SwitchCtree Core Dump Fixed (0410)**

FairCom corrected a potential for addressing a NULL pointer within SwitchCtree, which could cause an application to invoke an exception error.

**Windows 95 Standalone Multi-user Crash Resolved (1204)**

A problem that only affects Windows 95 applications using Standalone Multi-user libraries has been resolved. c-tree Plus includes dynamic detection of the Windows redirector version (vredir.vxd), in order to determine whether or not c-tree Plus must use I/O locking to prevent the redirector from caching data for network file I/O operations. The function that scans the vredir.vxd file for version information, which is only called when running on Windows 95 systems, may attempt to read past the end of a memory buffer, causing an access violation. This code has been adjusted to ensure this no longer happens.
13.3 Other Fixes

These fixes represent less serious or less likely issues corrected in recent builds.

**Shutdown Adjusted for Long Client Cleanup (1204)**

Under qualified circumstances it was possible for the Server to improperly terminate during shutdown. If a client thread required an exceptionally long time to exit (for example, in the middle of a very long transaction) after indications that the Server was shutting down, it was possible for the Server to free resources used by the client before the client completed processing. This caused an exception violation. This adjustment ensures that these excessively involved clients are given adequate time to finish before the Server terminates.

**Rebuild IEOF_ERR(519) Fixed (1204)**

A situation where the rebuild of an index containing resources returned IEOF_ERR(519) was corrected.

**Server Crash Fixed (1204)**

To correct a Server crash, an internal NULL check was applied to prevent an internal pointer with a NULL value from being used.

**NLM Invalid LOCAL_DIRECTORY Fix (1206)**

An issue related to an invalid or non-existent directory name being used with the LOCAL_DIRECTORY server keyword has been resolved. The c-tree Server assumes that when a directory name is specified for the LOCAL_DIRECTORY keyword, that that directory already exists and has proper permissions set in order for the Server to access this directory. Prior to this fix, the Server did not contain validation logic that verified the proper existence of this directory name. If an invalid directory name was specified, the Server would improperly terminate causing the Netware server to hang. A check has been added in order for the Server to validate this directory and properly terminate if invalid.

**Unexplained Delete and Unlock Errors Corrected (1231)**

A qualified uncommon scenario where record updates, deletes or unlocks resulted in an unexplained error has been addressed. Because of the resulting errors, this issue would be obvious if it affected an application. This issue was qualified to the use of fixed length records under transaction control when using any variation of the ctKEEP functionality (i.e., ctKEEP; ctKEEP_OUT; ctKEEP_OUT_ALL) during transaction Commit or Abort. Users who use the FREE locking mode for Commit or Abort are not affected by this problem. Record updates and deletes would most often experience a KDEL_ERR(4) error, while record un-lock calls would receive a UDLK_ERR(41).
Users who take advantage of the c-tree Plus automatic ISAM transaction support (i.e., SetOperationState(OPS_AUTOISAM_TRN)) should note that the transaction commits use ctKEEP or ctKEEP_OUT.

For fixed length data records, deleting the record under transaction control can lead to the actual lock on the deleted record being released but the user’s private lock table still considers the lock as owned by the user. This will occur if the commit attempts to ctKEEP or ctKEEP_OUT_ALL, or if no LockISAM state is turned on at delete time (say because OPS_LOCKON_GET was used to get a record lock) and a commit with ctKEEP_OUT is called. Then if this same user attempts to get a lock on this same record subsequently, the user lock table entry keeps the server from acquiring a system lock (because the system trusts the user lock table). For variable length records, a commit with ctKEEP or ctKEEP_OUT_ALL can lead to the same situation. This issue has now been resolved.

**KEEP_OUT_ALL Behavior with RestoreSavePoint (1231)**

If a lock is obtained on a fixed length record before a transaction, then after Begin and a SetSavePoint the record is deleted, and then the delete is undone by a call to RestoreSavePoint, the lock will remain. Prior to this fix, this lock had been freed. This change preserves the lock, as one would expect.

**DLOK_ERR on Record Add Corrected (1231)**

A subtle problem which could result in an AddRecord, AddVRecord, or NewData receiving a lock error (DLOK_ERR(42)) has been addressed. This situation could happen if one user issues a "put-to-sleep" (blocking) type of lock, and while waiting for the lock, the associated record is deleted by another user. Once the other user frees his lock, then the deleted record position (now placed in the deleted record stack available for re-use) is locked by the user who was waiting for a lock. Then, if a different user issues any form of add record, which calls NewData to obtain a new record position from the delete stack, this add record will fail with a DLOK_ERR(42) because this new record position is locked.

Although this situation seems common, it is actually highly unlikely, and will typically only arise under extreme conditions where the same record is constantly being locked (in blocking mode), deleted, and reused over and over.

**Server Shutdown with Many Clients Fix (0216)**

When a large number of clients (over 100) are attached to the c-tree Server, and the Server is shut down there were isolated occurrences where the Server would not give all clients adequate time to exit. This would cause the Server to core dump. Additional changes have been applied to the shutdown timing of the primary thread to avoid this situation.
**Windows Server Thread Exit Memory Leak (0216)**

A small memory leak has been fixed that affected both the c-tree Server and the Bound Server model. A potential leak of 116 bytes was reported for each terminated thread. Therefore, Servers that are continually servicing users logging on/off over a long period of time (without any Shutdown) should take note of this issue.

---

**c-tree Server Threading Issue on AIX 4.3 Resolved (0223)**

An issue in the Server internal defer logic on AIX 4.3 has been corrected. Users experiencing problems with Dynamic Dumps or Server shut down should consider upgrading.

---

**Obscure RestoreSavePoint Error (0314)**

An obscure issue which might have caused an unexpected I/O error such as RRED_ERR(407) after a RestoreSavePoint operation has been resolved. This is a moderately serious bug, but one not likely to occur. The following pseudocode sequence would lead to the problem:

```plaintext
Begin
  update record or resource
SetSavePoint
SetSavePoint
  update same record or resource
ClearSavePoint
RestoreSavePoint
```

After this sequence the first update will be gone as well as the second update, which was purposely undone by the RestoreSavePoint. In practice, virtually no one will perform this sequence explicitly. However, internally, when updating FairCom resources, we use SetSavePoint/ClearSavePoint to isolate our behind-the-scenes updates from the user’s explicit calls. This situation is most likely to occur, if at all, during a file creation sequence involving transaction dependent files in which the Begin is called before the file create call, the user calls SetSavePoint after the create, the user then calls something like PutDODA and then RestoreSavePoint. This leaves the resource header all FF’s which causes the RRED_ERR(407). This problem has been rectified.

---

**Windows 98 Shutdown Issue Resolved (0314)**

Prior to this fix, under Windows 98, if a c-tree Server was running in Tool-Tray mode and the user selected "Start Menu/Shutdown", the c-tree Server terminated but the Windows operating system did not finish its shutdown procedure. Internal adjustments were made to allow the c-tree Server to indicate to the operating system that it was okay to continue the shutdown process.
Win32 FPUTFGET Read Error Corrected (0314)

The default setting in c-tree Plus is to disable the I/O LOCKING that was previously enabled for all Win32 environments in the prior release, controlled by the macro ctPortWFW. c-tree Plus automatically enables the I/O LOCKING if the platform does not have the proper "redirector" (vredir) installed. Generally this code will only be enabled for Windows 95 or possibly a version of Windows 98 that doesn't have a current vredir installed.

In such a mixed environment (FPUTFGET) where one or more machines are doing I/O LOCKING and the others are not, there is a potential for a READ_ERR(36) error because the locks applied by the I/O LOCKING are to the actual data area of the file. Under DOS/WINDOWS, areas of a file locked are not even readable by another process, which produces the READ_ERR(36). In this situation the READ_ERR(36) can be viewed the same as a ITIM_ERR(160) error and the application needs to retry or delay and retry to perform the c-tree Plus function. If all clients are using the I/O LOCKING, then the retry is automatically handled inside the I/O LOCKING code.

In order to solve this problem, we applied a fix so that if a read or write fails and the system error code is ERROR_LOCK_VIOLATION (indicating an application that is not using I/O locking was locked out by an application that is using I/O locking), the non-I/O locking application attempts to get a lock and then retries the read or write operation if the lock was successful. After applying this fix, the READ_ERR(36) no longer occurred.

Dynamic Dump Recover with LOG_ODD/LOG_EVEN (0410)

Prior to this fix, dynamic dump recovery (ctrdmp) mishandled the LOG_EVEN and LOG_ODD keywords, returning LOPN_ERR(96).

Mac Multi-Threaded Standalone (0410)

A number of subtle issues related to implementing the Multi-threaded Standalone model on the Apple Macintosh have been cleaned up. Users experiencing any issues on the Mac with this model should update to this newer set of code.

DeleteIFile Not Deleting Superfile Member Indices (0410)

DeleteIFile now correctly removes the indices for superfile members. An error that could lead to only the data file being deleted has been corrected.

Standalone Multi-user Segmented File Error (0410)

The Standalone Multi-user model was not properly handling index file headers when activating new file segments, returning a SEGU_ERR(683). This model now requires EXCLUSIVE access to a file to activate new segments. This is not required in client/server models.
Incorrect Byte Swapping for CT_F2STRING and CT_F4STRING (0410)

In a heterogeneous environment, the bytes of CT_F2STRING or CT_F4STRING fields may have all been reversed instead of just the length fields preceding the strings. Now only the length fields in CT_2STRING, CT_F2STRING, CT_4STRING, and CT_F4STRING are reversed, not the entire string field.

Multi-Threaded Standalone Issues Resolved (0420)

Multi-Threaded Standalone issues on 64 bit platforms related to RegisterCtree, return type casting, and mutex structure sizes have been fixed. Users experiencing any type of instability in this model on 64 bit platforms should consider updating.

Eliminate terr’s Corresponding To Corrupt Indices (0424/0717)

A number of internal “terr” errors (terminating errors) have been adjusted within the c-tree Server. Prior to these adjustments, if a specific user (thread) encountered an internal integrity problem, the Server would shut itself down in what FairCom calls a “terr” situation. The terr errors addressed were 218, 220, 240, 7214, and 8214, which correspond to corrupt indices.

If a user thread encounters a bad index condition, the Server sends the application an error rather than terminating. These types of errors a rare and do not affect most users.

Dynamic Dump Restore FTYP_ERR(53) Corrected (0424)

When a file larger than 2GB was restored with the ctdmp utility, an FTYP_ERR(53) error was improperly generated. This was corrected.

Variable-Length Rewrite Serial Number Behavior Changed (0424)

c-tree Plus now rewrites variable-length records containing SRLSEG segments without incrementing the serial number in the file header.

Prior to this change a variable-length record rewrite that moved the record in the file incremented the internal serial number in the header of the file. The serial number in the record was not changed, but this caused a gap in the sequence of serial numbers being generated.

BAT_RET_REC With Huge Files On Mixed Client/Server (0424)

A huge file byte-flipping issue when using DoBatch with the BAT_RET_REC option has been repaired. The client-side code that performs the byte flipping did not account for the 8 byte versus 4 byte record position that precedes each record image. This was fixed by accounting for the record position entry based on the file’s huge attribute.
DoBatch with Heterogeneous Clients (0525)

An issue was corrected with DoBatch when using a key containing a numeric segment in a heterogeneous environment. TransformKey was not called, which sets numeric keys segments to be in the correct byte order by the time the key gets to the Server. Note: This may not have been a broad problem since most applications use the batch routines (or the set routines) with keys containing non-numeric segments.

Obscure Bad Node Split Issue Repaired (0525)

A unique scenario was discovered that could result in an improperly formed index that might produce a subsequent BIDX_ERR(527). This problem only occurs when:
1) A transaction controlled index holds unique keys;
2) The nodes may only have room for less than 8 key values;
3) An attempt is made to delete and re-add the same key at a different record position within the same transaction (e.g., rewriting a variable length record with an unchanged unique key, and the new record image does not fit in its original file position);
4) Either DECADD is explicitly chosen in a call to AddKey (not very likely), or in V7 the adaptive node split logic requests a DECADD node split. (DECADD implies keys are being added in decreasing key-order);
5) The deleted key value (still pending commit) and the re-added key value (still pending commit) occupy the first two positions in the leaf node.

If ALL of these conditions are met, the node split logic attempted to split the node before the first key value, which is a meaningless position within the node. This is a V6 and V7 issue although it is much more unlikely to occur in V6.

Client Hang Superfile Member RebuildIFile (0525)

An issue with a client application appearing to hang when calling RebuildIFile for a superfile member has been corrected. An OpenCtFileXtd call (performed to verify the superfile member exists) did not “or” in the ADMOPEN file mode bit. Without this bit, the OpenCtFileXtd call caused the Server to believe additional data will be returned by the RebuildIFile call. This causes the client and server to get out of synch in some communication environments (not Windows).

UNIFRMAT with Conditional Index (0525)

An internal issue has been resolved related to the use of Conditional Indexes with UNIFRMAT. Any user who might have used this configuration and bumped into any problems are encouraged to update to this latest set of code.
Server Shutdown if first COMM_PROTOCOL fails (0525)

An issue was resolved related to shutting down the server if the first COMM_PROTOCOL defined in the Server configuration file (ctsrvr.cfg) fails to load. When a communication protocol fails to load at startup, the Server issues a warning but continues using any other defined protocol. In the case of this error, when the request was made to shutdown the server, an error would be generated preventing a clean shutdown.

Additional DEBUG Information for 8521/8522 (0717)

In the event of a terr(8521) or terr(8522), additional information is output.

CTFLUSH Read-only Files Issue Resolved (0717)

A customer reported a problem while running from a CD. Because the files were “read-only” the customer was experiencing an FSAV_ERR(49) error when the CTFLUSH function was being called. This issue has been fixed.

Dynamic c-tree Plus DLL Switching Issues Resolved (0808)

A few issues related to the use of dynamic c-tree Plus DLLs with different Operational Models have been corrected. Users who experienced problems while trying to support simultaneous c-tree Plus models (e.g., Client and Standalone) in separate DLLs from the same application should benefit from these fixes.

Visual Basic 16 bit (0808)

A number of small syntax issues have been resolved related to 16 bit use with Visual Basic. Although this might be consider legacy, FairCom is happy to continue to support the models used by all our customers.

DODA field lengths for CT_INT8 and CT_INT8U (0808)

The CT_INT8 and CT_INT8U in a DODA behave the same as CT_INT4 and CT_INT4U when a DODA field length is left blank (zero). A DODA field length for CT_INT8 and CT_INT8U are now automatically set to 8 if their DODA length fields are left blank (set to zero).

TCP/IP – 64Bit ’localhost’ Issue (0816)

TCP/IP clients in 64-bit environments could only connect to c-tree Servers from the host machine, or localhost. An internal correction changed the client from treating the IP address as a 32-bit value, to a 64-bit value when appropriate, allowing a proper connection.

TCP/IP – Better Support for Many Simultaneous Logons (0816)

The c-tree Server now handles a larger number of simultaneous logon attempts without error.
CLIFIL and DELIFIL unresolved in Server SDK (0908)

The functions CloseFile and DeleteFile are no longer unresolved when using the c-tree Server SDK.

Borland v4 (BC4) and Pharlap DOS286 (0908)

At the request of a customer, FairCom made some minor corrections to provide support for Borland C v4 with the Pharlap DOS286 extender.

OPNRFIL with Deferred Close FUSE_ERR (0914)

FairCom corrected an issue that caused OpenFileWithResource to return FUSE_ERR when opening and closing a large number of files with deferred file close enabled: SetOperationState(OPS_DEFER_CLOSE, OPS_STATE_ON).

HP-UX 11 File Descriptor Limit (0914)

FairCom added logic to allow the c-tree Server for HP-UX 11 to increase its file descriptor limit up to the operating system’s hard file limit to avoid artificially limiting the number of files and connections available to the Server.

Cancel of Scheduled Dynamic Dump Crash Fixed (0918)

FairCom corrected an error that resulted in a Server crash when a scheduled dynamic dump was canceled.

End Of Notes