
A. Update Notes Version 6.8A

These update notes, combined with the existing documentation, provide the necessary detail to put FairCom's latest technology to work for you and your customers.

A.1 Introduction

Our recent technical efforts, which resulted in the current release of the **c-tree Plus**[®] and the **FairCom Server**, focused primarily on details reported by our customers. As you will see, a tremendous amount of work has been accomplished to resolve a considerable number of issues. Some issues involved production problems, while others are obscure. Regardless of its classification, if it happens to you, it is important.

This release of **c-tree Plus** adds Flexible File Limits, a streamlined application header, several new API functions, and other enhancements. See section A.2 for details and implementation instructions.

The **FairCom Server** now includes file encryption, expanded transaction processing capabilities, enhanced file mirror support, and other enhancements. See section A.3 for details and implementation instructions.

Section A.4, Miscellaneous Notes, lists minor adjustments and corrections that will not affect the majority of **c-tree Plus** developers.

Appendix B contains function description pages for all functions added in this release.

A.2 c-tree Plus: Features and Enhancements

Flexible File Limits

It is no longer necessary to know (up front) the total number of **c-tree Plus** index and data files that will be used in your program.

Before this release, you needed to set the *files* parameter used by **c-tree Plus** initialization routines, (e.g, **InitLSAM**, **InitCTree**, etc.), to cover the total number of files your application required. In many dynamically driven applications, this limit is not known, therefore developers were forced to set these numbers to (unreasonably) high limits to cover the possibilities. Of course, this utilized unnecessary resources, and was not a complete solution.

With this release, **c-tree Plus** allocates actual **c-tree Plus** FCBs only as needed for more efficient memory use. Internally, **c-tree Plus** considers the files parameter, *files*, to represent an initial block of file structures. Whenever the number of files required

by your application exceeds this initial amount, **c-tree Plus** automatically allocates another block of file structures using this number. No immediate mandatory modifications are needed to your applications for backward compatibility. However, for efficiency, applications initializing a large number of files should be changed to request a lower the number of files.

Resizing Issues with Flexible File Limits

Client file information, on both client and server side, is automatically resized when:

- A file open or create uses a file number beyond the existing client file range.
- A new file number is assigned with automatic file number assignment, (e.g., **OpenFileWithResource** with a -1 *filno*).

The FairCom Server keyword `MAX_FILES_PER_USER` defaults to 2048. An open/create which returns a `FINC_ERR(604)` implies that the create succeeded on the **FairCom Server**, but the client could not allocate memory for the local file info, and the newly created file has been closed.

Stand-alone applications support automatic resizing of file control information up to the limit imposed by `ctMAXFIL`, which defaults in *ctoptn.h* to 110. If `ctMAXFIL` is not defined, the limit defaults to 1024 files/FCBs.

NOTE: If a *filno* beyond the existing file range causes a resizing, the new number of files supported goes from 0 to $filno + MAXMEMB + 1$, with FCBs allocated for all potential file numbers in the range. Use automatic file number assignment for maximum memory efficiency.

For example, if **InitISAM** requests 100 files and **OpenCtFile** uses file number 1000, resizing changes the number of files supported to $1000 + MAXMEMB + 1$, or 1032. All the files between 100 and 1000 are now available. By contrast, if a automatic file number assignment causes resizing, the file number range is only extended by $MAXMEMB + 1$. If **InitISAM** requests 100 files and **OpenFileWithResource(-1,...)** causes resizing, the number of files supported would increase to 132.

New c-tree Plus Application Header

The new *ctreep.h* header file was introduced to accomplish two things:

Make c-tree Plus application development easier - This is the only **c-tree Plus** header file your application needs. No longer do you have to decipher which header file is needed and struggle with the question "Do I Need *ctoptn.h*? ...*ctstdr.h*? ...*cterrc.h*? ...etc.?".

Global Variables - **c-tree Plus** offers a handful of global variables useful for error detection: *uerr_cod*, *sysiocod*, *isam_err*, and *isam_fil*. Currently, in most environments (like Win32), **c-tree Plus** defaults to a `ctNOGLOBS` configuration. When `ctNOGLOBS` is defined, these variables must be address through a pointer to

the current **c-tree Plus** instance, (*ctWNGV*). If this is not handled properly, these variables may show up as unresolved within your application, or your compile will fail stating that *ctWNGV* is not defined. By including *ctreep.h*, the internals are handled for you. If you depend on these variables, simply include this new header file, and its internals will automatically handle the proper references whether you are configured for `ctNOGLOALS` or not.

See our sample programs and *ctreep.h* for more information.

LockISAM mode returns current lock state

A new mode, `GETLKISAM`, was added to **LockISAM**. It is designed to give you a way to query the current ISAM lock state. **LockISAM**(`GETLKISAM`) returns the current ISAM lock state as a negative value, as shown below:

| Symbolic Constant | ISAM lock state | GETLKISAM return |
|-------------------|-----------------|------------------------------|
| FREE | 0 | 0 (no current lock state) |
| RESET | 1 | -1 |
| ENABLE | 2 | -2 |
| ENABLE_BLK | 3 | -3 |
| READREC | 4 | -4 |
| SUSPEND | 5 | -5 |
| RESTORE | 6 | -6 |
| RESTRED | 7 | -7 |
| RESTORE_BLK | 8 | -8 |
| READREC_BLK | 13 | -13 |
| RESTREC_BLK | 14 | -14 |

If no error occurs, **LockISAM**(`GETLKISAM`) sets *isam_err* to the return value. An error during **LockISAM**(`GETLKISAM`), returns a positive value corresponding to the error code, *isam_err*.

See the **LockISAM** function description in the **c-tree Plus Programmers Reference Guide** for more details on the ISAM lock state.

RenameFile and RenameFileXtd Added

RenameFile and **RenameFileXtd** atomically rename some or all of the files associated with the IFIL structure *ifilptr*, and update the internal IFIL resource.

RenameIFile

Atomically rename some or all of the files associated with the IFIL structure.

SHORT NAME RENAMEIFIL

TYPE ISAM function

DECLARATION `COUNT RenameIFile(ifilptr)`
 `pIFIL ifilptr;`

DESCRIPTION **RenameIFile** atomically renames some or all of the files associated with the IFIL structure *ifilptr*, and updates the internal IFIL resource. *ifilptr* must point to a complete IFIL structure corresponding to an exclusively opened IFIL, with the names replaced by the new names. If the "new" name is the same as the original name, no renaming takes place for that file. The *tfilno* member of *ifilptr* must contain the file number of the data file associated with the IFIL. The *dataextn* and *indxextn* parameters can be used to modify the file name suffixes of the data and/or index files.

If the data file supports transaction logging, two criteria must be satisfied:

- No transaction can be active when **RenameIFile** is called; and
- The files cannot be updated between the exclusive open and the renaming call.

If either of the criteria is violated, **RenameIFile** returns `TEXS_ERR(70)` or `FCRP_ERR(14)`, respectively.

The renaming and optional updating of the IFIL resource are performed atomically under the control of a transaction automatically managed by **c-tree Plus**.

The data file's IFIL resource will be updated under the following conditions:

- Resources have been enabled.
- The `DISABLERES` bit of the *dfilmod* member of *ifilptr* is not on.

When the IFIL resource is updated, then the following protocol is used:

- If an existing IFIL resource is found, then only the name fields are updated to reflect the renaming operations.
- If no existing IFIL resource is found, then the IFIL information passed into **RenameIFile** is used in its entirety.

The IFIL structure passed into **RenameIFile** must agree with both the physical files and the existing IFIL resource (if any) on the following points:

- The number of indices,
- the key length and duplicate key status of each index, and
- the specification of which indices serve as a host index, keeping in mind that the *aidxnam* member of `IIDX` can create host indices in addition to the "base" index.

If any of these characteristics do not match, **RenameFile** returns `IAIX_ERR(608)`.

The first index in the IFIL may derive its name either from the data file name, or through a non-NULL *aidxnam* parameter. The new IFIL may change whether or not the *aidxnam* parameter is used for the first index. It cannot change whether *aidxnam* is used for any of the other indices.

RETURN

| Value | Symbolic Constant | Explanation |
|-------|-----------------------|---|
| 0 | <code>NO_ERROR</code> | Successful rename of file. |
| 14 | <code>FCRP_ERR</code> | File updated since <code>EXCLUSIVE</code> open. |
| 70 | <code>TEXS_ERR</code> | Transaction in progress. |
| 608 | <code>IAIX_ERR</code> | IFIL resources too different. |

See Appendix A of the **c-tree Plus Programmer's Reference Guide** for a complete listing of valid **c-tree Plus** error values.

EXAMPLE

```
extern IFIL    myfile;
             COUNT    retval;

if (retval = InitISAM(6,7,4))
    printf("\nCould not initialize. Error %d.", retval);
else {
    if (OpenRFile(-1, "MyFile.dat", EXCLUSIVE))
        printf("\nCould not open files.");
    else {
        if (RenameIFile(&myfile))
            printf("Could not rename IFIL, error %d.",
                isam_err);
    }
    if (CloseISAM())
        printf("\nCould not close ISAM.");
}
```

LIMITATIONS

If the data file supports transaction logging, two criteria must be satisfied:

- No transaction can be active when **RenameFile** is called; and
- The files cannot be updated after the exclusive open and before the renaming call.

The IFIL structure passed into **RenameFile** must agree with both the physical files and the existing IFIL resource (if any) on the following points:

- The number of indices,
- the key length and duplicate key status of each index, and
- the specification of which indices serve as a host index, keeping in mind that the *aidxnam* member of IIDX can create host indices in addition to the "base" index.

SEE ALSO

RenameFileXtd and **RenameFile**.

RenamFileXtd

Rename ISAM files, extended version.

SHORT NAME RENIFILX

TYPE Extended ISAM function

DECLARATION

```
COUNT RenamIFileXtd(ifilptr, dataextn, indxextn)
pIFIL    ifilptr;
pTEXT    dataextn, indxextn;
```

DESCRIPTION **RenamFileXtd** is a variation of **RenamFile** that permits the file extensions to be changed. This section expands on the description of **RenamFile**.

dataextn and *indxextn* point to buffers specifying the data and index file name extensions, respectively. The extensions are 8 byte ASCIIZ (NULL terminated ASCII) strings. If the pointers are NULL, the default extension will be used: ".dat" for data files and ".idx" for index files. For files with no extension, pass a pointer to a buffer that contains only blanks terminated by a NULL character.

RETURN As with **RenamFile**. See Appendix A of the **c-tree Plus Programmer's Reference Guide** for a complete listing of valid **c-tree Plus** error values.

TestFileNbr

A new API function, **TestFileNbr**(*filno*), determines the status of a file number.

SHORT NAME TSTFILNUM

TYPE Low-level function

DECLARATION

```
COUNT TestFileNbr(filno)
COUNT    filno;
```

DESCRIPTION **TestFileNbr** returns zero if *filno* is not in use or returns FINT_ERR (**c-tree Plus** not initialized), FNUM_ERR (*filno* is out of range), or FUSE_ERR (*filno* is in use). **TestFileNbr** DOES NOT SET OR CHANGE *uerr_cod*.

RETURN

| Value | Symbolic Constant | Explanation |
|-------|-------------------|------------------------------------|
| 0 | NO_ERROR | File number not in use. |
| 22 | FNUM_ERR | <i>filno</i> is out of range |
| 46 | FUSE_ERR | <i>filno</i> is in use |
| 47 | FINT_ERR | c-tree Plus not initialized |

See Appendix A of the **c-tree Plus Programmer's Reference Guide** for a complete listing of valid **c-tree Plus** error values.

EXAMPLE

```

COUNT   retval;

if (retval = InitISAM(6,7,4))
    printf("\nCould not initialize. Error %d.", retval);
else {
    if ((retval = TestFileNbr(5)) == 0) {
        if (OpenRFile(5, "MyFile.dat"))
            printf("\nCould not open files.");
    } else if (retval == 46)
        if (OpenRFile(-1, "MyFile.dat"))
            printf("\nCould not open files.");
    if (CloseISAM())
        printf("\nCould not close ISAM.");
}

```

New ClearSavePoint Function

A new savepoint clear function, **ClearSavePoint**, removes the most recent savepoint WITHOUT UNDOING the changes made since the savepoint. **ClearSavePoint** puts pre-image space in the same state as if the most recent savepoint had never been created.

ClearSavePoint

Remove the last savepoint without undoing other changes.

SHORT NAME

SAVPCLR

TYPE

Low-level function

DECLARATION

```
COUNT ClearSavePoint()
```

DESCRIPTION

ClearSavePoint removes a savepoint WITHOUT UNDOING the changes made since the savepoint. **ClearSavePoint** puts pre-image space in the same state as if the most recent savepoint had never been created. By comparison, **RestoreSavePoint** cancels changes made since the last savepoint, but does NOT remove the savepoint. **ClearSavePoint** takes no arguments, and returns an error code if it fails, or zero if successful.

RETURN

| Value | Symbolic Constant | Explanation |
|-------|-------------------|---------------------------------|
| 0 | NO_ERROR | Savepoint cleared. |
| 71 | TNON_ERR | No savepoint or no transaction. |

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

RestoreSavePoint, SetSavePoint.

Run time check of feature support via SystemConfiguration

To permit an application to check features supported within **c-tree Plus**, the **SystemConfiguration** return array has a new element providing a feature bit map. When a particular bit is turned on in this element, *cfgFEACHK* of the **SystemConfiguration** output array, the corresponding feature is available. This is useful for client applications where the availability of a feature is not known at compile/link time. The current feature bit mask definitions are:

| | |
|----------------------|--|
| cfgFEACHKplusName | +index name support for PermlIndex . |
| cfgFEACHKCLOSEInTran | Close or delete of an updated file inside an active transaction does not abort the transaction: either an error is returned, or the close or delete is deferred until transaction commit or abort. |
| cfgFEACHKencryption | Encryption is supported. |
| cfgFEACHKflexFile | Flexible file limits are supported. |

For example, to check if the new "+name" index file naming convention for use with **PermlIndex** or **TemplIndexXtd** is supported, see the following calls:

```
LONG sysinfo[ctCFGLMT];

InitISAM(...);
SystemConfiguration(sysinfo);
if (sysinfo[cfgFEACHK] & cfgFEACHKplusName)
    printf("\n+name convention is supported\n");
```

NOTE: An initial **c-tree Plus** call (**InitCtree**, **InitCtreeXtd**, **InitISAM**, etc.) must be made before **SystemConfiguration** can check on the *cfgFEACHK* element.

IFIL file name and location enhancement

This feature permits index files created by **PermlIndex** or **TemplIndexXtd** to be automatically located in the same directory as the associated data file. With **PermlIndex** only, it also allows the IFIL entry for the index file to have the same directory as the associated data file, even if the data file is not in its original directory.

To activate this feature, the name of an index file should be of the form:
+index_name

By placing a plus sign, '+', as the first character of the index name, the new index automatically "follows" the data file. In other words, it is created in the same directory as the data file. The second point above regarding **PermlIndex** and the IFIL entry is more significant than it may seem. **c-tree Plus** can now open files via **OpenFileWithResource** in a directory different from the original directory which appears in the internal IFIL resource. When this happens, **OpenFileWithResource** expects any index that had the same directory as the data file to be in new directory.

Note: The plus sign is not part of the actual file name used to create the index. Mirrored file name components express this option independently. For example, the following would all be "legal" for aidxnam entries:

```
+primary_index|+mirror_index  
+primary_index|mirror_index  
primary_index|+mirror_index
```

Only the name components beginning with the plus sign invoke this new feature.

Path work for OpenFileWithResource

We changed the **OpenFileWithResource** automatic index path adjustment, and corrected the index path adjustment with mirrored files. It has been modified so if a data and index file share the same path, any adjustment made to the data path is made to the index path, including the removal of a path on the data file. If an index file originally has no path, then a path adjustment will be applied to the index file only if the data file originally had no path.

Windows 16-bit auto instance enhanced

Older programs using **InitCtreeGV** now work properly with `ctPORTAUTOWIN`.

For absolute integrity under 16-bit Windows, the **c-tree Plus** focus, *ctWNGV*, should be established before each **c-tree Plus** function call. Because we felt this was a bit tedious for the **c-tree Plus** programmer, the feature `ctPORTAUTOWIN` was added a few years ago. Starting with **c-tree Plus v6.7**, `ctPORTAUTOWIN` was activated as the default. Developers who controlled their own *ctWNGV* through **InitCtreeGV** were required to de-activate this default by placing `#undef ctPORTAUTOWIN` in *ctcmpl.h*. This is no longer the case. The **c-tree Plus** library supports auto instances, auto-instance control, or user control (**SetCtreeGV**, **GetCtreeGV**).

Rebuild and Compact Enhanced

A variety of changes and features in this release related to file rebuilds. This section discusses these enhancements.

Multiple Rebuilds

Performing multiple, simultaneous rebuilds with the **FairCom Server** resulted in conflicting sort-work file names. This was resolved by changing how the thread ID was used in generating the file names. Files with the same file name, in different directories, can also rebuild simultaneously.

Improved scan

Rebuild and compact now use an improved scan for valid variable-length record mark. When rebuild finds an invalid record mark, it scans for a valid one. This check now includes a comparison of the size of the file and the total length in the potential record header.

Compact opens corrupt files

The compact utility and functions can now open corrupt files, allowing corrupt files to be compacted without first being rebuilt.

New return codes for RebuildFile

With the addition of the new rebuild purge technology, **FairCom Update Guide V6.7, section 2.5.11.F1, pg. 61**, when **RebuildFile** encounters unexpected duplicate keys and/or bad serial numbers, it can behave in three different ways:

- If `purgeIFIL` is used and a binary stream file can be opened, the duplicate keys and bad serial numbers are automatically purged from the indices and the data records are deleted from the file. A **RebuildFile** returns `DUPJ_ERR(650)`, and the temporary stream file contains the deleted data records.
- If `purgeIFIL` is NOT used and an ASCII stream file can be opened, the duplicate keys and bad serial numbers are not added to the indices, but the data records remain in the file. **RebuildFile** returns `DUPL_ERR(652)`, and lists the offending keys and data records in the stream file. BEFORE THIS MODIFICATION, REBUILDIFILE RETURNED `NO_ERROR(0)` UNDER THIS CONDITION.
- If a stream file cannot be opened, **RebuildFile** returns either `KDUP_ERR(2)` or `KSRL_ERR(605)` when duplicate keys and/or bad serial numbers are encountered. FURTHER, NO MORE INDICES ARE PROCESSED. That is, the rebuild does not run to completion as it does in the above two cases.

GETFIL enhanced

GETFIL now allows a query of the corrupt (update) flag in a file's header. The `UPDFLG` mode was added to return the current value of the update flag. **GETFIL** returns the current value of the update flag as a `LONG` integer. File headers contain a one-byte update flag indicating the corrupt status of the file:

| | |
|------|--|
| 0x00 | OK |
| 0x4c | index must be rebuilt: loop found in leaf level node links |
| 0x52 | opened with <code>OPENCRPT</code> and the file was corrupt |
| 0x63 | file was compacted and indices must be rebuilt |
| 0xff | corrupt |

Unix exclusive file lock more efficient

The default methods for locking are improved in multi-user non-server mode. **c-tree Plus** defaults to a more efficient Unix exclusive file lock. The previous method of exclusively locking a Unix file, so that other **c-tree Plus** applications could not open the file, was locking the entire file. Non-exclusive opens lock a single byte in a user lock region. Locking the entire file appears to be very inefficient, especially with NFS mounted file systems. The new default locks only the user lock region on an exclusive open. The disadvantage of the new approach is that non-**ctree Plus** applications can

update the file opened in exclusive mode within **c-tree Plus**. The new default is invoked with `#define ctLOCK_EXCLUSIVE`. To use the previous approach, use `#define ctLOCK_EXCLUSIVEfile`.

OS/2 DLL Support Added

Borland C TCP/IP Client Support via Visual Age DLL added – Because TCP/IP support is not readily available for OS/2 Borland C v2.0 users, **c-tree Plus** now allows a Borland C application to use a **c-tree Plus** TCP/IP DLL compiled and linked with Visual Age. Complete TCP/IP support is available via the IBM's Developer Connection for Visual Age. To accomplish this, the "pascal" calling conventions are used both when compiling the DLL with Visual Age (`#define ctDECL _Pascal` in *ctree\source\cset.os2\ctcmpl.h*), and when compiling the application with Borland C v2.0 (`#define ctDECL __pascal` in *ctree\source\borlandc.os2\ctcmpl.h*).

Two higher level `#defines` have been added to activate the definitions found in *ctcmpl.h*. Add `#define ctPortBORLANDC_COMPATIBLE_DLL` to your *ctoptn.h* when compiling a Visual Age DLL. Add `#define ctPortUSE_VISUALAGE_DLL` to your Borland *ctoptn.h* when building your application.

Because of the `_pascal` naming convention for Visual Age, you may find some functions from the .DEF file coming up unresolved when linking the DLL. This is a case sensitivity problem. Edit the .DEF file (*ctree\source\ctclient.def*). Change the following function names to upper case.

| | |
|------------|-----------|
| ctdidx | cttseg |
| frmkey | ctuseg |
| ctcdelm | ctRENFIL |
| reset_cur | setfndval |
| cttestfunc | |

CTUNF1 improved

CTUNF1 was updated to support 8-byte alignment and Security Resource conversion. These utilities correctly process administrative resources in files created by the **FairCom Server**. The resource holding permission mask and other security information was not correctly transformed by *CTUNF1* because the encryption key generation was not modified for `LOW_HIGH/HIGH_LOW` differences. *ctunf1.c* has been updated to support correct transformation. A separate, small module, *ctunfs.c*, suitable for distribution in object form, contains the confidential routines to decrypt and re-encrypt the resource.

The need for *ctunfs.c* is controlled by `ctUNFMserver`. If it is defined, then *ctunfs.c* is required and the sign-on banner indicates that transformation of client/server files is supported. If it is not defined, then the banner does not mention client/server files and the first security resource encountered during transformation will generate a warning to use the executable *CTUNF1* distributed by FairCom.

Expression Parser

In **c-tree Plus** v6.7, FairCom introduced a powerful new feature, Conditional Index Support. This feature can be viewed as a sophisticated index level filter determining whether a given record is indexed based on a C language syntax expression.

At the heart of Conditional Index Support is an expression parser/analyzer capable of parsing NULL-terminated string expressions based on the C language syntax. The expression parser recognizes numeric and string constants, DODA field names, predefined functions such as `strcmp()` and `atoi()`, and standard mathematical and Boolean operators.

Even if you choose not to use full Conditional Index Support, you can use the expression parser/analyzer to evaluate expressions, as described below. For a complete sample program, see `ctexpr.c` in the `../ctree/source` directory.

Using **c-tree Plus's** expression parser/analyzer involves two steps, described in detail below:

1. Calling **`cndxparse()`** to parse your expression, producing an expression tree the expression analyzer can evaluate.
2. Calling **`cndxeval()`** to evaluate the expression tree using data from a buffer in memory.

Parsing your expression

Parsing your expression involves three steps:

- a) Define a DODA structure.
- b) Parse the DODA into a record schema and field name list.
- c) Parse your expression to produce an expression tree.

Sample code to perform these steps is shown below. This code assumes **c-tree Plus** has been initialized prior to calling **`ctparsedoda()`**.

```
#include "ctcndx.h" /* For PTREE type */

/* Define a DODA structure. */
DATOBJ doda[] = {
    {"CustomerNumber", 0, CT_INT4U},
    {"ZipCode", 4, CT_FSTRING, 9},
    {"State", 13, CT_FSTRING, 2},
    {"LastName", 15, CT_STRING, 37},
    {"FirstName", 52, CT_STRING, 37},
    {"Address", 89, CT_STRING, 49},
    {"City", 138, CT_STRING, 37}
};

COUNT retval; /* Return code. */
PTEXT schema; /* Record schema. */
PTEXT names; /* Field name list. */
```

```

PTREE ptree; /* Expression tree. */
PTEXT expr; /* Expression string. */
/* Parse the DODA into a record schema and field name list. */
if ((retval = ctparsedoda(doda, 7, &schema, &names)) != 0)
    printf("Error %d parsing DODA.\n", retval);

/* Parse your expression to produce an expression tree. */
expr = "strcmp(LastName, \"Smith\") == 0
      && CustomerNumber > 10000";
ptree = cndxparse(schema, names, expr, strlen(expr));
if (!ptree)
    printf("Error: Unable to parse expression.\n");
else
    printf("Successfully parsed expression.\n");

```

To invoke the expression parser to parse a string expression and produce an expression tree, call **cndxparse()**, which is declared as follows:

```

PTREE cndxparse(Schema, Names, InputText, InputTextSize)
    PConvMap Schema;
    PTEXT Names;
    PTEXT InputText;
    NINT InputTextSize;

```

Schema is a pointer to a record schema derived from a DODA definition you provide. *Names* is a pointer to a list of the field names from the DODA. *InputText* points to a NULL-terminated string expression, and *InputTextSize* is the length of *InputText*.

One of the most useful features of the expression parser is its ability to associate symbolic names in expressions with data in a buffer in memory. In order to use this ability, you must define a record schema, known as a DODA (Data Object Definition Array). A DODA is an array of field specifications, each of which contains a field name, field offset, field type, and a field length. By providing the expression parser with a DODA, you may include references to DODA field names in your expressions. See the sample code shown later in this section for an example of a DODA definition.

While a DODA is conveniently defined in your application using an array of DATOBJ structures, **cndxparse()** does not take a DODA in DATOBJ form, but instead accepts a record schema and a list of the field names from the DODA. In order to simplify converting your DODA into the required record schema and field name list, FairCom has written a utility function, **ctparsedoda()**. This function can be found in the sample file *ctexpr.c*. **ctparsedoda()** is declared as follows:

```

COUNT ctparsedoda(doda, numfld, ppschema, ppnames)
    PDATOBJ doda;
    UCOUNT numfld;
    ppTEXT ppschema;
    ppTEXT ppnames;

```

Evaluating your expression

Having produced an expression tree, you are ready to evaluate the expression using the expression analyzer. To evaluate your expression, call **cndxeval()**, which is declared as follows:

```
COUNT cndxeval(Tree, Recptr, Schema)
    PTREE      Tree;
    pVOID      Recptr;
    pConvMap   Schema;
```

Tree is the expression tree returned by **cndxparse()**. *Recptr* points to a buffer containing data you wish to evaluate with the expression. *Schema* is the record schema **ctparsedoda()** produced from your DODA definition. The record schema is used to associate the data in *Recptr* with field names specified in your expression.

NOTE: Before you call **cndxeval()** the first time, you must ensure that a run-time stack has been allocated for the expression analyzer.

To summarize, evaluating your expression involves three steps:

- a) Allocate a run-time stack for the expression analyzer (first time only).
- b) Set up a buffer containing the field data used in the expression.
- c) Evaluate the expression.

If you wish, you can repeat steps b) and c) multiple times. Sample code to perform these steps is shown below. It is assumed the `Get_Buffer()` routine allocates a record buffer and initializes it with data conforming to the field definitions specified in the DODA.

```
COUNT retcidx; /* Result of expression evaluation. */
PTEXT recbuf; /* Record buffer. */

/* Allocate a run-time stack for the expression analyzer (first
time only). */
if (!ctcidxStk) {
    ctcidxStk = (pVOID) getcndxmem(CNDX_MAX_STACK *
                                   ctSIZE(PLEAF));

    if (!ctcidxStk) {
        printf("Unable to allocate memory for run-time
              stack.\n");
        ctrt_exit(1);
    }
}

/* Set up a buffer containing the field data used in the
expression. */
Get_Buffer(&recbuf);

/* Evaluate the expression. */
retcidx = cndxeval(ptree, recbuf, (pConvMap)schema);
```

```
if (retcidx<0)
    printf("The expression cannot be evaluated for this record
    - error %d.\n", uerr_cod);
else if (retcidx)
    printf("The expression evaluates to TRUE for this
    record.\n");
else
    printf("The expression evaluates to FALSE for this
    record.\n");
```

Remember to free the memory used by your record schema, field name list, and expression tree when you are finished using them. Use the following **c-tree Plus** functions to do so:

```
mbfree(schema);
mbfree(names);
cndxfree(ptree);
```

A.3 FairCom Server: New Features

This release of the **FairCom® Server** offers a year's worth of improvement since our last worldwide commercial release of V6.07.28. Here we present the issues addressed which make this the strongest, most proven FairCom Server to date.

Encryption

Introduction

FairCom Server V6.08.30 introduces encryption of data, index and transaction log files. This technology provides the means to add an extra level of confidentiality to an application's data. Once encrypted, it becomes impossible for a casual user to "dump" or "inspect" the data.

The following sections describe the details.

Basic Encryption

FairCom's default level of encryption may be thought of as a simple camouflage mode protecting data from unauthorized end users.

To encode the log files place `LOG_ENCRYPT YES` in the server configuration file. The default is `NO`.

In order for a **c-tree Plus** client application to support encryption, you must decide which files will be encrypted. This decision must be made when creating a file. To encode index and data files without a parameter file, use **SetEncryption**, as below, before the file creation calls.

```
SetEncryption(pTEXT mod,pTEXT key,VRLen keylen)
```

mod is not used at this time, and should be `NULL` or point to a null-terminated ASCII string. *key* points to a byte array containing the encryption key, of length *keylen*. **SetEncryption** does NOT assume that *key* points to a null-terminated ASCII string. The key can be any arbitrary array of bytes. Key lengths of seven or more should be adequate. To stop encrypting new files, call **SetEncryption** with *key* set to `NULL` and/or *keylen* set to zero.

SetEncryption only affects file creation operations. All files created after a given call to **SetEncryption**, with a non-`NULL` *key* and a *keylen* greater than zero, will be encrypted with the same key. Therefore, at the ISAM level, a data file and its associated indices will be created with the same encryption key. Turning encryption on and off through calls to **SetEncryption** only affects whether new files are encrypted. Once a file is set for encryption, it is always encrypted.

The following pseudo-code causes the first ISAM data file and its indices to be encrypted, and the second ISAM data file and its indices not to be encrypted:

```
InitISAM(...)  
SetEncryption (NULL,key,(VRLEN) 23)  
CreateIFile(..1..)  
SetEncryption (NULL,NULL,(VRLEN) 0)  
CreateIFile(..2..)
```

To encode index and data files with a parameter file, use the new user profile bit, `USERPRF_ENCRYPT`, in **CreatelSAMXtd** to enable encryption. All files created by, and after, **CreatelSAMXtd** will be encrypted with the same key, selected automatically by **c-tree Plus**. New data and index files created after a call to **SetEncryption** with a `NULL` *key* and/or zero *keylen* are not encrypted.

SetEncryption does NOT affect transaction log file encryption. Log file encryption is controlled in the **FairCom Server** configuration file, as described above.

Two new error codes have been defined: `DCOD_ERR(606)` is returned when an application cannot decode an encrypted file. `RCOD_ERR(607)` is returned when automatic recovery cannot decode an encrypted file.

Some of the **c-tree Plus** utilities (e.g., *CTRBLD*) use a very low-level open and header read sequence. Code has been added to permit the opening sequence to be followed by an extraction of the encryption key, which happens automatically at the higher open levels. Support has also been added to permit dynamic dump/recovery to support encrypted files and logs.

Advanced Encryption

To add an additional layer of encryption protection, an application can have a "hidden" key, which modifies the "public" key. Any application linked with the hidden key will not be able to read an encrypted file. To use the hidden key, `#define ctCAMOSys` must be included in *ctoptn.h*, and the module *ctsysk.c* compiled and added to the library for the **FairCom Server**. *ctsysk.c* contains a short function which stores the "hidden" key. The **FairCom Server SDK** provides the modules and libraries required to implement this additional level of encryption. This ensures only **c-tree Plus** applications linked with your customized FairCom Server can access the encrypted data.

SetEncryption

Set or reset encryption key.

SHORT NAME ctSETENCRYPT

TYPE Low-level function

DECLARATION

```
COUNT SetEncryption(mod, key, keylen)
pTEXT    mod, key;
VRLEN    keylen;
```

DESCRIPTION

To encode index and data files without a parameter file, use **SetEncryption** before the create file calls. *mod* should be NULL or point to a null-terminated ASCII string. *mod* is not used at this time. *key* points to a byte array which comprises the encryption key, of length *keylen*. **SetEncryption** does NOT assume that *key* points to a null-terminated ASCII string. The key can be any arbitrary array of bytes. Key lengths of seven or more should be adequate. To stop encrypting new files, call **SetEncryption** with *key* set to NULL and/or *keylen* set to zero.

SetEncryption only affects file creation operations. All files created after a given call to **SetEncryption**, with a non-NULL *key* and a *keylen* greater than zero, will be encrypted with the same key. Therefore, at the ISAM level, a data file and its associated indices will be created with the same encryption key. Turning encryption on and off through calls to **SetEncryption** only affects whether or not a new file is encrypted. Once a file is set for encryption, it is always encrypted.

The following pseudo-code encrypts the first ISAM data file and its indices, and does not encrypt the second ISAM data file and its indices:

```
InitISAM(...)
SetEncryption (NULL,key,(VRLEN) 23)
CreateIFile(..1..)
SetEncryption (NULL,NULL,(VRLEN) 0)
CreateIFile(..2..)
```

SetEncryption does NOT affect transaction log file encryption.

RETURN

| Value | Symbolic Constant | Explanation |
|-------|-------------------|--|
| 0 | NO_ERROR | Successful operation. |
| 82 | UALC_ERR | No memory available to allocate. |
| 454 | NSUP_ERR | ctCAMO not defined. Service not supported. |

See Appendix A of the **c-tree Plus Programmer's Reference Guide** for a complete listing of valid **c-tree Plus** error values.

Connections or Users?

The **FairCom Server** is licensed on a concurrent connection basis. Each connection to the **FairCom Server** counts against the total until the connection is closed. A client establishes a connection with every successful **c-tree Plus** initialization call, (e.g., **InitCtreeXtd**, **InitISAMXtd**, etc.), with multi-threaded application potentially establishing many connections. Close connections with **StopUser** or **CloseISAM**.

At startup, the **FairCom Server** displays the current connection limit. The default, 25, can be changed with **CONNECTIONS** keyword in *ctsrvr.cfg*. This replaces the **USERS** keyword, which is still support for backward compatibility. If **CONNECTIONS** exceeds the licensed limit of the **FairCom Server**, the licensed limit is used.

Server Configuration File Comments

Individual lines in *ctsrvr.cfg* can now be commented out with either a semi-colon ';' or a slash '/' in front of a line. This is a small, but helpful, change. Example:

```
COMM_PROTOCOL FSHAREMM  
;COMM_PROTOCOL F_TCPIP  
/FUNCTION_MONITOR YES  
MEMORY_MONITOR 200000
```

The configuration file parser expects two tokens per line. Do not use this feature to add extended comments to the middle of the configuration file. It is intended to easily comment out standard keywords.

Transaction Processing Enhanced

Transaction Active Log Adjustment.

It is desirable to minimize the number of active log files, and the associated disk space usage, when exceptional circumstances have caused an increase in the number of active log files. Extensive logic has been added to automatically determine when log files are no longer needed. No action is required to take advantage of this feature.

c-tree Plus and the **FairCom Server** dynamically determine the oldest log file required for recovery with the benefits:

- 1) Awareness of how many active logs are no longer required and can be deleted.
- 2) When the flushing strategy does not keep pace, the number of active logs can automatically increase to safeguard automatic recovery.

Closing Files During Transaction Processing

Before this release, when a process tried to a close a file that the process had updated under transaction control, but which was not committed or aborted, the transaction was automatically aborted and the file closed. The current default behavior fails the

attempted close with error CPND_ERR(588). To return to the previous default behavior, add `#define ctBEHAV_AbortOnCLOSE` to *ctoptn.h*. For the **FairCom Server**, add the following to the **FairCom Server** configuration file:

```
COMPATIBILITY ABORT_ON_CLOSE
```

Note: This behavior also applies to a file DELETE under the same circumstances.

Optional defer of CLOSE until transaction commit/abort

SetOperationState can change the behavior of a file close request when the file has been updated as part of a still active transaction. Turning on `OPS_DEFER_CLOSE` causes defers a file close or delete until the transaction is committed or aborted.

Updates by other clients do not affect the "update status" of a file for a client who has not updated the file within a transaction.

For example:

| | |
|--|-----------|
| OPEN FILE A | client #1 |
| OPEN FILE A | client #2 |
| OPEN FILE B | client #2 |
| SETOPS(OPS_DEFER_CLOSE,OPS_STATE_ON); | client #1 |
| SETOPS(OPS_DEFER_CLOSE,OPS_STATE_ON); | client #2 |
| TRANBEG | client #1 |
| TRANBEG | client #2 |
| READ FILE A | client #1 |
| UPDATE FILE A | client #2 |
| CLOSE FILE A (succeeds without defer) | client #1 |
| CLOSE FILE A (deferred) | client #2 |
| UPDATE FILE B (succeeds without defer) | client #2 |
| TRANEND (causes a CLOSE ON FILE A) | client #2 |
| CLOSE FILE B | client #2 |
| TRANEND | client #1 |

This example shows that client #2's update does not cause a defer when client #1 closes file A within a transaction in which client #1 had not updated file A, but client #2's close is deferred.

- A request to reopen the file with the same file number within the same transaction will be honored.
- An attempt to reopen the file with a different file number within the same transaction will be honored, provided there is no overlap with the original file numbers due to index members.
- An attempt to open a different file with the same file number or overlapping file numbers will fail.

For example:

```
OPEN A as file #2 (where A has two additional index members)
OPEN B as file #10
SETOPS(OPS_DEFER_CLOSE,OPS_STATE_ON)
TRANBEG
UPDATE A
CLOSE A (deferred: file #s 2,3 and 4 still in use by A)
UPDATE B
OPEN A as file #2 (succeeds since same file reusing file #'s)
UPDATE A
CLOSE A (deferred: file #s 2,3 and 4 still in use by A)
UPDATE B
OPEN C as file #2 (fails because of defer on file A)
OPEN A as file #0 (fails because of overlap with itself)
OPEN A as file #9 (fails because of overlap with B)
OPEN A as file #5 (succeeds)
OPEN C as file #2 (succeeds because file A now reassigned)
```

There are restrictions on file mode changes between a deferred close and a subsequent reopen within the same transaction. A file originally opened in exclusive mode can be reopened in SHARED mode. A file originally opened in SHARED mode must be reopened in SHARED mode. READFIL is not allowed on a reopen.

NOTES:

- Superfile hosts are not affected by turning on OPS_DEFER_CLOSE.
- **Avoid accessing a file whose close is deferred pending a transaction commit/abort.** Generally, attempts from a client side application return FACS_ERR(26). Attempts by a single-user transaction processing application may succeed. **FairCom does not specify or guarantee the actual results of accessing a file whose close is deferred prior to reopening the file.**

Defer File Delete

If OPS_DEFER_CLOSE is on, a file delete on a file updated by a still active transaction will be deferred until the commit or abort of the transaction. An attempt to reopen the file within the same transaction results in a FNOP_ERR(12) as if the file were actually gone. An attempt to re-create the file within the same transaction will not succeed because the file has not yet been deleted.

Long Transactions

This release affects long transactions that would be abandoned or forced to abort, but for which the increase in number of active log files permits them to continue. With the default setting, FIXED_LOG_SIZE OFF, the number of active log files increases to allow the transaction to continue. If FIXED_LOG_SIZE is on, the transaction is abandoned with error TABN_ERR(78). This applies to both single-user and client/server applications.

MIRROR Enhancements

The **FairCom Server** and stand-alone **c-tree Plus** applications offer support for file mirroring that is very easy to implement. By simply defining a file to be mirrored, along with the mirror name, **c-tree Plus** and the **FairCom Server** maintain an exact duplicate file. In this release, we have addressed a number of issues related to mirrors.

Automatic copy of mirrors

Mirrored files are now automatically copied during rebuild, automatic recovery and dynamic dump restore. The out of sync member of a mirrored pair of files is automatically copied over at the conclusion of automatic recovery. Automatic recovery failed to process mirror, or primary, files before this modification only when one of them was out of sync with the other.

Better Recovery File Survival on Failure

Before this modification, the **FairCom Server** could be forced to stop unexpectedly when one of the mirrored start files and/or mirrored log failed during a save, open, or create operation.

Ordinarily, a log write or read failure to one of the primary/mirrored pair would be handled without failure by the **FairCom Server**. The code to open, create, or save log and start files has been modified to permit the **FairCom Server** to continue operation if one of the pair of files fails. The **FairCom Server** attempts to delete an existing mirrored start file if the primary file was newly created. It attempts to rename an existing mirrored log if the primary log file is newly created. To revert to the old behavior, add the following to the **FairCom Server** configuration file:

```
COMPATIBILITY LOG_MIRROR_FAILURE
```

NOTE: At startup, the primary and mirrored log and start files must all be working.

Advanced Mirrored File Work

The **FairCom Server** now accommodates re-acquisition of a lost, mirrored start file. Before this modification, if a primary or mirror of a mirrored pair of start files became inaccessible, and then subsequently accessible again, a `MHDR_ERR(549)` could occur during a checkpoint, causing the **FairCom Server** to terminate with a `L37 ccatend`. The modification stores the "lost" mirror status, so that a potential `MHDR_ERR` can be by-passed if due to re-acquisition of the lost member of the start file mirrored pair.

SKIP_MISSING_LOG_MIRRORS

We now accommodate missing start/log file mirrors at startup. A new **FairCom Server** keyword, `SKIP_MISSING_LOG_MIRRORS`, which defaults to `NO`, has been added. If `SKIP_MISSING_LOG_MIRRORS YES` is in `ctsrvr.cfg`, the **FairCom Server** starts up even if start file and/or log file mirrors are missing or out of sync. In the case of out of sync start files the most current start file is used. If a mirrored start or log file is missing, the **FairCom Server** still starts up. If the log files are out of sync, the

PRIMARY LOG FILE is used in all cases. The **FairCom Server** will not start up if the mirrored log appears better than the primary log when SKIP_MISSING_LOG_MIRRORS is turned on. The **FairCom Server** ignores a mirrored log unless it matches the primary log. The **FairCom Server** determines which of the two logs is more current. If the mirrored log appears more current, the **FairCom Server** startup fails with LPR1_ERR(664).

FairCom Unix Server Shutdown and <CTRL><C>

The **FairCom Server** ignores <CTRL><C>. Adding the following keyword to the *ctsrvr.cfg* permits <CTRL><C> to stop the **FairCom Server**.

```
CONSOLE CONTROL_C_ENABLE
```

FairCom Server Shutdown Issues

We improved the shutdown sequence in many ways. Please note the following:

Shutdown Final Checkpoint

When beginning the shutdown sequence, the **FairCom Server** attempts to shut down each client thread in an orderly manner. This modification skips the **FairCom Server's** final system checkpoint when some clients do not terminate, to:

- Cause an automatic recovery on the next startup to clean up any undone transactions.
- Avoid conflicts and possible core dumps between the **FairCom Server** shutting down and the still existing clients. This is especially likely for pre-image transactions/files. If the final system checkpoint is skipped, a message to this effect is entered in *CTSTATUS.FCS*, and the shutdown sequence sent to the monitor also indicates the skip.

Shutdown Delay

A **FairCom Server** shutdown delay was added to allow a large number of clients to disconnect. To help minimize the chance for clients to be active when the **FairCom Server** is shutting down, a longer defer loop has been added to give clients a chance to detect a shutdown and begin their orderly disconnect. Previously, the **FairCom Server** was not allowing enough time for disconnect when a large number of clients (> 200) were attached.

CT_USER Enhancement

Considerable adjustments have been made that make it easier for developers to use the CT_USER feature within the **FairCom Server SDK**. Now, it is easy to configure your code as a module to load dynamically or to link statically into your customized **FairCom Server**.

Diagnostics keyword added to track files at logon and logoff

When the `DIAGNOSTICS FILE_LOGON` keyword is added to the **FairCom Server** configuration file, upon each logon and logoff, four file counters are output: physical files open, logical files open, File Control Blocks (FCBs) in use, and FCBs available. The logical files count will be greater than physical files if superfiles are in use. FCBs in use count will be greater than logical files if index files contain additional index members. These values reflect counts generated by all applications using the **FairCom Server**.

Dynamic Dump Status information added

Progress reports are now available in `CTSTATUS.FCS`. The optional `DIAGNOSTICS` argument, `DYNDUMP_LOG`, generates dynamic dump status info, sending progress entries during each dynamic dump to `CTSTATUS.FCS`, including an entry for each file it attempts to dump.

Platform-Specific Enhancements

FairCom Windows Server Enhancements

- **NO_MESSAGEBOX Server Keyword** - Added a new **FairCom Server** keyword, `CONSOLE NO_MESSAGEBOX`, which deactivates error messages coming to the console in the form of a message box. The **FairCom Server** continues to log all messages to `CTSTATUS.FCS`. This feature was added to prevent console messages when the **FairCom Server** is used as a Windows NT Service.
- **Performance Change** - Users reported a change in performance from the last release because of our new `COMMIT_DELAY` default of 1. We changed the default to (-1) for FairCom Windows NT Servers. This deactivates the feature by default. Users may still use the `COMMIT_DELAY` server configuration keyword to override the default.
- **NETBIOS console message cleanup** - Because the **FairCom Server** automatically attempts to listen on all available LANAs for NETBIOS, misleading communication messages would display on the **FairCom Server's** console if a particular LANa was not alive. These unnecessary messages were removed.
- **Windows NT Service interface enhanced** - The Windows NT Service interface now uses the system registry. Startup and shutdown have been improved.
- **Shared Memory Adjustment** - The internal communications structure has been properly padded so the alignment of the client is no longer an issue.
- **Win32 SPX Speed Enhancement** - Adjusted the ECB Size parameter in `at_SessionSend` to prevent unnecessary network overhead. This improves SPX network performance.

FairCom Solaris Server Adjustments

We received a number of reports of various problems with the **FairCom Server** running on Solaris. These issues were typically found to be operating system dependent and have been addressed. Because of a few isolated issues that could cause the **FairCom Server** to hang or crash, we strongly recommend all users on Solaris upgrade to this newest release. We would like to extend a special thanks to those customers who presented FairCom with the information on these items.

- **Defer logic** - A modification was made within the **FairCom Server** defer logic relating to an 'absolute time' value being used which caused the timed-wait semaphore function parameter to overflow, resulting in a **FairCom Server** termination.
- **Native Thread Launch** - A memory problem was fixed in the native thread launch code that could cause a problem at **FairCom Server** startup time.
- **File Descriptor Issues** - The **FairCom Server** internal call to increase the file descriptors available to the **FairCom Server** has been increased to 1024 if the current system's value is found to be less.
- **Streamed file descriptors** - Problems with the dynamic dump, the *CTSTATUS.FCS* log and various other operations occurred when a large number of files were opened (over 255). The problem related to the fact that after 255 descriptors were used for **c-tree Plus** files, any attempt to use a file handle above 255 failed on any stream operation (e.g.: Dynamic Dump tries to open its script file). This was isolated to Solaris, and workaround code has been applied.

Apple Computer, Inc.

FairCom remains committed to both current and future operating environments from Apple Computer, Inc. The following examines issues related to Apple:

- **File Encryption** - All FairCom Mac Servers support data, index, and transaction log encryption. FairCom Server SDK developers may either implement FairCom's default encryption algorithms or, for an extra level of security, supply their own encryption algorithms.
- **MacIPX Introduced** - To improve Apple client performance with the FairCom NLM Server, the SPX communication protocol has been implemented on the Mac. As Novell has recently acknowledged, its TCP/IP implementation under Netware was quite slow (up to an 80 millisecond delay to get data to the wire). Customers supporting Mac clients against an NLM Server should evaluate this enhancement.
- **68K Mac Server Problem** - Users of the 68K version of the FairCom Mac Server labeled 6.07.28c(Build 0802) or less should update to the latest version. The #define PERC_D was used during generation instead to the #define PERC_HD. This caused invalid scanning of parameter files.

- **CodeWarrior Pro 3** - The latest version of **c-tree Plus** has been tested with the latest Metrowerks compiler. Complete new project files are included.
- **Mac Server Zone Can Be Specified** - ADSP clients can now specify a zone for a **FairCom Server** connection. Pass `ServerName@ZoneName` in the initialization function.
- **Mac Server name syntax problem fixed** - The use of colon in the **FairCom Server** name was not supported. For many platforms, the run-time protocol can be specified by the delimiter of '^' or ':'. BUT, for the Mac, the ':' is used to delimit the "Server Name" from the "Object Type", as:

```
ServerName:FairComServer@ZoneName
```

Therefore, the Mac does not check here for the ':' and protocols can only be specified for the Mac via a '^', for example:

```
FairComServer^TCPIP
```

FairCom NLM Server

- **SPX Abend Repaired** - When an application terminated, the **FairCom Server** proceeded to time-out and disconnect the server side thread for that user. If, at the same time, the system administrator issued a kill user request, the NLM abended.
- **TCP/IP Speed** - For quite some time, customers have asked about disappointing performance results with the TCP/IP protocol support for the Netware NLM implementation of the **FairCom Server**. After numerous evaluations, FairCom's engineers were convinced that it was the result of a weakness within Novell's TCP/IP layer, not FairCom's implementation. Although we could never get Novell to recognize this, we are happy to see that Novell is now publishing patches to its TCP/IP stack on NetWare. For more information, see <http://www.support.novell.com>.

FairCom AIX 4.3 Server

- The FairCom RS/6000 Server now supports either the native AIX pthread implementation or the DCE pthread standard. We have experienced memory leaks within the DCE thread implementation (within IBM's code, not ours) as well as limitations to the number of threads that can be supported (255). Therefore we are pleased that the newest version of AIX V4.3 apparently implements pthreads, and has a very large thread limit.

A.4 Miscellaneous Notes

c-tree Plus

- **GETCURP/GETCURK did not clear isam_err upon success** - **GETCURP** and **GETCURK** did not clear the *isam_err* to zero on a successful call. Internal calls to these routines are now made to **iGETCURP/iGETCURK**, respectively. These functions behave as before (i.e., not affecting *isam_err*). The revised **GETCURP/GETCURK** now set *isam_err* appropriately.
- **DUPCHANNEL Efficiency Modification** - When a DUPCHANNEL open request fails, **c-tree Plus** closes the extra file channels opened before the failure. This change also sets the extra descriptor fields to an invalid value.
- **NO_VARLD support** - This release cleaned up syntax errors when no variable-length record support is used. All references to variable-length routines when variable-length record support is disabled were removed. When using **NO_VARLD** you must also define **NO_RESOURCE** and **NO_SUPER**. Resources and superfiles require variable-length support.
- **Added an option to cause CTSTATUS.FCS to stay open** - Prior to this modification, *CTSTATUS.FCS* was opened, written to, and closed for each status log entry. If `#define ctBEHAV_StatusOpen` is included, then the status files is opened only once, and each entry causes a **fflush()** call. This avoids a problem in those environments which assume that stream files will be opened with a small file descriptor (e.g., less than 256).
 - If the `#define` is not active, then the behavior is as before.
 - If the **CTSTATUS_SIZE** keyword is used, each time a new log extent is created, the status file must be closed and reopened. Avoid the **CTSTATUS_SIZE** option in systems prone to stream file descriptor limitations.
- **RES_LENGTH return mode added to GetCtResource when searching by RES_POS** - The **RES_LENGTH** mode in **GetCtResource** specifies the return of an array of three **LONGS**: the resource type, the resource number and the resource length. Prior to this modification, **RES_LENGTH** was not available when the resource was retrieved by absolute position using the **RES_POS** mode.
- **Optional #define ctUSE_FCALLOC added to 16-bit Windows** - This `#define` controls the type of **alloc** used by **Windows: GlobalAlloc** or **_falloc**. **c-tree Plus** uses **GlobalAlloc** by default for 16-bit Windows memory allocation. Customers requested **_falloc** be available for special implementations, such as **COM**, so that memory is not freed when the parent application terminates. Adding `#define ctUSE_FALLOC` in *ctsw16_w.c* sets the default from **GlobalAlloc** to **_falloc**. See `#define ctUSE_FCALLOC` in *ctsw16_w.c* for details.

- **UNIFRMAT Alternate Collating Sequence** - We corrected byte ordering for the Alternate Collating Sequence component of IFIL with UNIFRMAT.
- **Conditional Index** - We corrected a pointer arithmetic error in the conditional index lexical scan.
- **Removed Obsolete Error Code** - `SKTY_ERR` doesn't exist and was removed.
- **Extended locks with 4GB files** - The computation of the byte to lock in extended locking required a cast to avoid a signed arithmetic problem when the index/data position moves beyond the 2GB level.
- **Internal Integrity Verification Added** - If a `CT_STRING` field is not delimited, the specified field maximum length will be used in the `ctconvert2()` field scan. While we always expect a `CT_STRING` field to be delimited, as opposed to a `CT_FSTRING`, this modification helps avoid problems if the field is not delimited.
- **NEWID utility program fixed** - Because the definitions of `BUPDATE` moved from `ctcmpl.h` to `ctsfio_a.c` between v6.6 and v6.7, the *NEWID* utility program found in the `techhelp` directory was modified to compile and work properly.
- **PRMIIDX pfilnam Pointer Integrity Check** - The **PRMIIDX** documentation indicates that only a few members of the IFIL structure must be initialized. Before this release, the client-side processing for **PRMIIDX** checked for a valid file name, because the checking is performed by a multi-purpose routine, even though the name is not used. A NULL name causes a `BIFL_ERR(198)`. This release uses a local IFIL structure with name set to ensure no preprocessing problem on the client-side.
- **BORLANDC bigadr fix** - Added appropriate BorlandC necessities to the 16-bit 'bigadr' function.
- **VOLATILE symbols changed to ctVOLATILE** to avoid C++ header conflict.
- **Corrected core on non-PROTOTYPE systems** - The internal `ntlogoff` call had a parameter syntax error for non-`PROTOTYPE` platforms resulting in a core dump. This only applies to clients with `PROTOTYPE` turned off.
- **LOADKEY repaired** - Corrected an error where a non-server application calling **LOADKEY** could cause an index buffer used by the high-speed load to be reassigned without the high-speed load recognizing the reassignment. This could lead to `terr(233)` or `terr(235)`. The application program performed a large number of index operations between successive calls to **LOADKEY**.
- **Corrected GPF or 7491 error when using strcmp in conditional index** - Modified the code used to parse and evaluate the conditional expression.

- **Added support for BSD Unix locking.**
- **Mac NULL conflict with C++ adjusted.**
- **Internal IFIL integrity check added** - c-tree Plus now checks IFIL/PARM info for agreement with actual index attributes.
- **Corrected Windows 16-bit REGCTREE problem** - Corrected an error in 16-bit Windows allowing linked lists of registered c-tree Plus instances to become corrupt.
- **Microsoft C v6.0 cleanup** - Added prototype definitions for MSC60. FairCom does not formally support communications with this older compiler, however, some prototype adjustments were made to make life easier for developers who chose to support MSC60 on their own.
- **Borland C v5.0 Header file conflict** - Changed the `NOwait` (Case Sensitive) defined in `ctcomm.h` to avoid a conflict with **Borland 5**.
- **FRCKEY: Bad file number and % out of range caused problem** - Two error conditions, bad file number or percentage out of range, caused **FRCKEY** to incorrectly return `uerr_cod` instead of `DRNZERO`. The fix simply ensures `uerr_cod` is set, but **FRCKEY** returns `DRNZERO`. There was also a cosmetic change to **ESTKEY** causing a return of 0 instead of a return of `DRNZERO`.
- **RNGENT and ESTKEY:** The single entry-point interface for **RNGENT** and **ESTKEY** calls **GETFIL** before invoking the actual **ctree()** call for **RNGENT** and **ESTKEY**. Before this release, **ctPREFNC** was called before **GETFIL**, which does its own **ctPREFNC**, resulting in two successive calls to serialize **ctree()** access. This caused `terr(1101)`. The modification calls **GETFIL** before **ctPREFNC**.
- **Virtual file close logic, (vtclose), improved under Win32** - The file creation logic under Win32 was not properly monitoring the `ERROR_TOO_MANY_OPEN_FILES` return from **CreateFile**. `vtclose` no longer virtually closes a file to give a file descriptor to the new, but uncreated, file.
- **ctPATH_SEP #defined to appropriate value in etcmpl.h**
- **Conditional Index string compare fixed** - The ignore case character compare returned the wrong result when the source character was less than the destination character.
- **PRMIDX, RBLIDX, and TMPIDXX return code** - Corrected error codes returned from these functions.
- **Corrected infinite loop in a superfile member under 'physical order' traversal**
- **Support low-level locks to force Microsoft I/O cache under FPUTFGET**

- **Solaris File Descriptor Support added to StandAlone c-tree** - StandAlone Solaris applications now automatically increase the available file descriptor limit. Change the default value of file descriptors, 1024, in `ctsint_a.c`.
- **Multi-Threaded Stand-Alone sleep** - Adjusted nano-second logic for stand-alone RS 6000, HP9000, Digital Unix, Solaris x86, and Solaris Sparc.
- **Single User Transaction Abort Logic adjusted** - Logic that detects that a file to be deleted has not been updated is now fixed.

FairCom Server

- **Corrected server core dump** when **CREMEM** was called with a member number greater than the number of members specified in the preceding **CREIDX**.
- **Corrected problem with ctdefer (ctsysnap)** - Corrected **ctdefer** in native-threaded **FairCom Server** in environments using **nanosleep()** to implement **ctdefer**.
- **TCP/IP Low-level retry Read/Write: Developer-defined retries** - In response to a customer request, a very low-level TCP/IP retry loop has been added to the read and write logic at the system network level. This retry option is controlled with the `ctTRYREAD` and `ctTRYWRITE` defines at the top of `./ntree/sockets/ctnlib.c`. This option should be used only by advanced users who have reviewed the TCP/IP code in `ctnlib.c` and understand the performance impact of the invoked retry.
- **TCP/IP Buffer size increased** - The default size of the TCP/IP communications buffer is increased from 2048 to 4096, except on 16-bit platforms.
- **ctRENFIL modified** - Prior to the addition of **RenameFile**, **ctRENFIL** performed the rename and made a log entry for `TRNLOG` files. A `TRNLOG` file could be renamed in an existing transaction as long as the file had not been updated. Now the rename is recoverable and can be undone during recovery or rollback. For `TRNLOG` files, there cannot be an active transaction when **ctRENFIL** is called, and the file cannot be updated prior to the rename operation.
- **CTSTOP changes** - **CTSTOP** has the following changes:
 - 1) The order of user prompts for ADMIN Name and Password changed.
 - 2) The User and Password can be passed from the command line.
 - 3) The delay option has been deactivated. It caused the client to appear to hang for the length of the **FairCom Server** shutdown delay. The delay option will be reactivated when this issue is resolved.
- **(SHARED | READFIL) file mode** - If a file is opened with a mode including both the `SHARED` and `READFIL` bits, then the **FairCom Server** would not allow the second and subsequent opens to succeed. The **FairCom Server** resolves the issue by clearing the `SHARED` bit if both are set.

- **Dynamic Dump Sleep Adjustment** - The dynamic dump was adjusted to allow a sleeping dump thread can be interrupted. The previous behavior, using **ctdefer** to sleep the thread, can be invoked at compile time by using `#define ctBEHAV_DumpDefer`. This is necessary in systems where a timed semaphore is not accurate with respect to the system clock (e.g., Banyan).
- **Corrected a ctcattend L64 error** with `PREIMAGE_DUMP YES` in *ctsrvr.cfg*, and a superfile directory index receives an incorrect file handle.
- **Corrected a potential deadlock** when a file was created, or a superfile member deleted, when a dynamic dump was starting up.
- **Windows 16bit LOCLIB/TCPIP compile error with MSVC cleaned up.**
- **Server L59 Shutdown Error when Pre-Image Files are in use solved.**
- **Fixed a deadlock condition in node cleanup with a large number of clients.**
- **Fixed Server recovery L65 error with the use of Pre-Image Files** - Resolved an L65 error that could occur during automatic recovery when `PREIMG` files are used inside a transaction.
- **Mirror/FCB issue** - Fixed a problem where an index member File Control Block, (FCB), was contaminated by leftover mirrored file info.
- **Corrected LOCLIB check for multi-threaded clients.**
- **Corrected Client-side IFIL conversion issue on platforms with 8-byte pointers.**
- **16-bit Windows LOCLIB:** Repaired declaration problems for 16-bit LOCLIB.
- **Corrected ctcattend L59 error due to a sign-extension system dependency.**

B. New Function Description Pages

This section provides detailed function description pages for c-tree Plus API functions added with this release. In a future release, these pages will be inserted into their proper positions in the **c-tree Plus Function Reference Guide**.

The following functions have been added:

| | |
|-----------------------|--|
| ClearSavePoint | Remove the last savepoint without undoing other changes. |
| RenameFile | Atomically rename some or all of the files associated with the IFIL structure. |
| RenameFileXtd | Rename ISAM files, extended version. |
| SetEncryption | Set or reset encryption key. |
| TestFileNbr | Determine the status of a file number. |

ClearSavePoint

Remove the last savepoint without undoing other changes.

SHORT NAME SAVPCLR

TYPE Low-level function

DECLARATION COUNT **ClearSavePoint**()

DESCRIPTION **ClearSavePoint** removes a savepoint WITHOUT UNDOING the changes made since the savepoint. Calling **ClearSavePoint** puts pre-image space in the same state as if the most recently called savepoint had never been called. By comparison, **RestoreSavePoint** cancels changes made since the last savepoint, but does NOT remove this savepoint. **ClearSavePoint** takes no arguments, and returns an error code if it fails, or zero if successful.

RETURN

| Value | Symbolic Constant | Explanation |
|-------|-------------------|---------------------------------|
| 0 | NO_ERROR | Savepoint cleared. |
| 71 | TNON_ERR | No savepoint or no transaction. |

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 **RestoreSavePoint, SetSavePoint.**

RenameIFile

Atomically rename some or all of the files associated with the IFIL structure.

SHORT NAME RENAMEIFIL

TYPE ISAM function

DECLARATION `COUNT RenameIFile(ifilptr)
 pIFIL ifilptr;`

DESCRIPTION **RenameIFile** atomically renames some or all of the files associated with the IFIL structure *ifilptr*, and updates the internal IFIL resource. *ifilptr* must point to a complete IFIL structure corresponding to an exclusively opened IFIL, with the names replaced by the new names. If the "new" name is the same as the original name, no renaming takes place for that file. The *tfilno* member of *ifilptr* must contain the file number of the data file associated with the IFIL. The *dataextn* and *indxextn* parameters can be used to modify the file name suffixes of the data and/or index files.

If the data file supports transaction logging, two criteria must be satisfied:

- No transaction can be active when **RenameIFile** is called; and
- The files cannot be updated between the exclusive open and the renaming call.

If either of the criteria is violated, **RenameIFile** returns `TEXS_ERR(70)` or `FCRP_ERR(14)`, respectively.

The renaming and optional updating of the IFIL resource are performed atomically under the control of a transaction automatically managed by **c-tree Plus**.

The data file's IFIL resource will be updated under the following conditions:

- Resources have been enabled.
- The `DISABLERES` bit of the *dfilmod* member of *ifilptr* is not on.

When the IFIL resource is updated, then the following protocol is used:

- If an existing IFIL resource is found, then only the name fields are updated to reflect the renaming operations.
- If no existing IFIL resource is found, then the IFIL information passed into **RenameIFile** is used in its entirety.

The IFIL structure passed into **RenameIFile** must agree with both the physical files and the existing IFIL resource (if any) on the following points:

- The number of indices,
- the key length and duplicate key status of each index, and
- the specification of which indices serve as a host index, keeping in mind that the *aidxnam* member of `IIDX` can create host indices in addition to the "base" index.

If any of these characteristics do not match, **RenameIFile** returns `IAIX_ERR(608)`.

The first index in the IFIL may derive its name either from the data file name, or through a non-NULL *aidxnam* parameter. The new IFIL may change whether or not the *aidxnam* parameter is used for the first index. It cannot change whether *aidxnam* is used for any of the other indices.

RETURN

| Value | Symbolic Constant | Explanation |
|-------|-------------------|------------------------------------|
| 0 | NO_ERROR | Successful rename of file. |
| 14 | FCRP_ERR | File updated since EXCLUSIVE open. |
| 70 | TEXS_ERR | Transaction in progress. |
| 608 | IAIX_ERR | IFIL resources too different. |

See Appendix A of the **c-tree Plus Programmer's Reference Guide** for a complete listing of valid **c-tree Plus** error values.

EXAMPLE

```
extern IFIL    myfile;
             COUNT    retval;

if (retval = InitISAM(6,7,4))
    printf("\nCould not initialize. Error %d.", retval);
else {
    if (OpenRFile(-1, "MyFile.dat", EXCLUSIVE))
        printf("\nCould not open files.");
    else {
        if (RenameIFile(&myfile))
            printf("Could not rename IFIL, error %d.",
                isam_err);
    }
    if (CloseISAM())
        printf("\nCould not close ISAM.");
}
```

LIMITATIONS

If the data file supports transaction logging, two criteria must be satisfied:

- No transaction can be active when **RenameIFile** is called; and
- The files cannot be updated after the exclusive open and before the renaming call.

The IFIL structure passed into **RenameIFile** must agree with both the physical files and the existing IFIL resource (if any) on the following points:

- The number of indices,
- the key length and duplicate key status of each index, and
- the specification of which indices serve as a host index, keeping in mind that the *aidxnam* member of IIDX can create host indices in addition to the "base" index.

SEE ALSO

RenameFileXtd and **RenameFile**.

RenameIFileXtd

Rename ISAM files, extended version.

SHORT NAME RENAMEFILX

TYPE Extended ISAM function

DECLARATION COUNT **RenameIFileXtd**(ifilptr, dataextn, indxextn)
 pIFIL ifilptr;
 pTEXT dataextn, indxextn;

DESCRIPTION **RenameIFileXtd** is a variation of **RenameIFile** that permits the file extensions to be changed. This section expands on the description of **RenameIFile**.

dataextn and *indxextn* point to buffers specifying the data and index file name extensions, respectively. The extensions are 8 byte ASCIIZ (NULL terminated ASCII) strings. If the pointers are NULL, the default extension will be used: ".dat" for data files and ".idx" for index files. For files with no extension, pass a pointer to a buffer that contains only blanks terminated by a NULL character.

RETURN As with **RenameIFile**. See Appendix A of the **c-tree Plus Programmer's Reference Guide** for a complete listing of valid **c-tree Plus** error values.

SetEncryption

Set or reset encryption key.

SHORT NAME

ctSETENCRYPT

TYPE

Low-level function

DECLARATION

```
COUNT SetEncryption(mod, key, keylen)  
pTEXT    mod, key;  
VRLEN    keylen;
```

DESCRIPTION

To encode index and data files without a parameter file, use **SetEncryption** before the create file calls. *mod* should be NULL or point to a null-terminated ASCII string. *mod* is not used at this time. *key* points to a byte array which comprises the encryption key, of length *keylen*. **SetEncryption** does NOT assume that *key* points to a null-terminated ASCII string. The key can be any arbitrary array of bytes. Key lengths of seven or more should be adequate. To stop encrypting new files, call **SetEncryption** with *key* set to NULL and/or *keylen* set to zero.

SetEncryption only affects file creation operations. All files created after a given call to **SetEncryption**, with a non-NULL *key* and a *keylen* greater than zero, will be encrypted with the same key. Therefore, at the ISAM level, a data file and its associated indices will be created with the same encryption key. Turning encryption on and off through calls to **SetEncryption** only affects whether or not a new file is encrypted. Once a file is set for encryption, it is always encrypted.

The following pseudo-code encrypts the first ISAM data file and its indices, and does not encrypt the second ISAM data file and its indices:

```
InitISAM(...)  
SetEncryption (NULL,key,(VRLEN) 23)  
CreateIFile(..1..)  
SetEncryption (NULL,NULL,(VRLEN) 0)  
CreateIFile(..2..)
```

SetEncryption does NOT affect transaction log file encryption.

RETURN

| Value | Symbolic Constant | Explanation |
|-------|-------------------|--|
| 0 | NO_ERROR | Successful operation. |
| 82 | UALC_ERR | No memory available to allocate. |
| 454 | NSUP_ERR | ctCAMO not defined. Service not supported. |

See Appendix A of the **c-tree Plus Programmer's Reference Guide** for a complete listing of valid **c-tree Plus** error values.

TestFileNbr

Determine the status of a file number.

SHORT NAME TSTFILNUM

TYPE Low-level function

DECLARATION

```
COUNT TestFileNbr(filno)
COUNT filno;
```

DESCRIPTION **TestFileNbr** returns zero if *filno* is not in use or returns FINT_ERR (**c-tree Plus** not initialized), FNUM_ERR (*filno* is out of range), or FUSE_ERR (*filno* is in use). **TestFileNbr** DOES NOT SET OR CHANGE *uerr_cod*.

RETURN

| Value | Symbolic Constant | Explanation |
|-------|-------------------|------------------------------------|
| 0 | NO_ERROR | File number not in use. |
| 22 | FNUM_ERR | <i>filno</i> is out of range |
| 46 | FUSE_ERR | <i>filno</i> is in use |
| 47 | FINT_ERR | c-tree Plus not initialized |

See Appendix A of the **c-tree Plus Programmer's Reference Guide** for a complete listing of valid **c-tree Plus** error values.

EXAMPLE

```

COUNT   retval;

if (retval = InitISAM(6,7,4))
    printf("\nCould not initialize. Error %d.", retval);
else {
    if ((retval = TestFileNbr(5)) == 0) {
        if (OpenRFile(5, "MyFile.dat"))
            printf("\nCould not open files.");
        {
        else if (retval == 46)
            if (OpenRFile(-1, "MyFile.dat"))
                printf("\nCould not open files.");
        if (CloseISAM())
            printf("\nCould not close ISAM.");
        }
    }
}

```