test link

c-treeACE V9.1.0

Release Notes

Audience

Developers

Subject

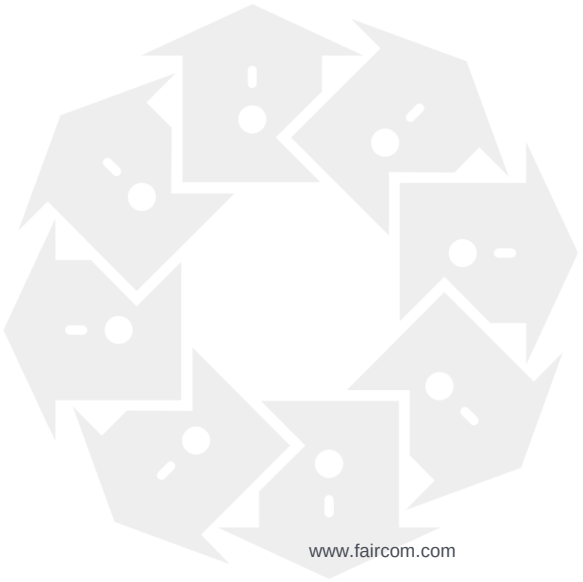
FairCom's high-performance NAV and SQL database technology.

FairCom®

Contents

1. Version V9.1.1 Updates and Changes

Since the release of Version 9.0, there have been additional enhancements, critical production updates, and modifications to c-treeACE that are included in this latest delivery. Please review the information in this document regarding these latest changes.

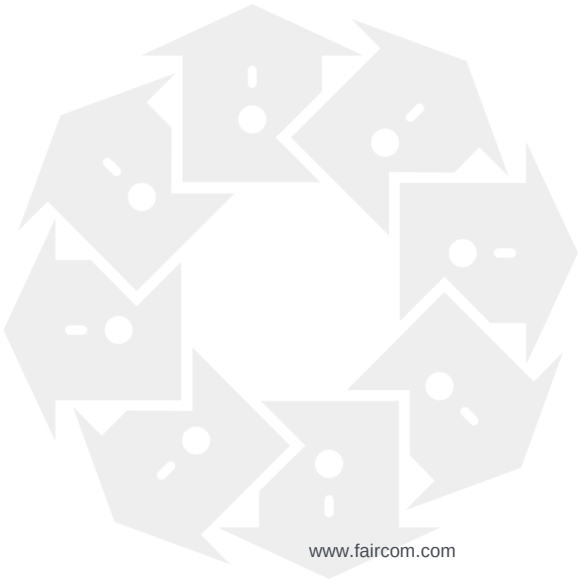


1.1 FairCom Typographical Conventions

Before you begin using this guide, be sure to review the relevant terms and typographical conventions used in the documentation.

The following formatted items identify special information.

Formatting convention	Type of Information
Bold	Used to emphasize a point or for variable expressions such as parameters
CAPITALS	Names of keys on the keyboard. For example, SHIFT, CTRL, or ALT+F4
FairCom Terminology	FairCom technology term
FunctionName()	c-treeACE Function name
Parameter	c-treeACE Function Parameter
`Code Example`	Code example or Command line usage
utility	c-treeACE executable or utility
filename	c-treeACE file or path name
`CONFIGURATION KEYWORD`	c-treeACE Configuration Keyword
CTREE_ERR	c-treeACE Error Code



2. Critical Production Updates

The following notifications have been found to be important considerations in many production systems. Please Contact your nearest Faircom office if you are concerned if any of these may impact your application or have any questions.

2.1 Potential c-tree Server Automatic Recovery Failures with the LOGIDX Feature

c-treeACE provides a feature called *LOGIDX* which can allow for quicker automatic recovery time on server startup should you experience an unexpected failure (for example, a power outage). This feature can be enabled with a file mode of *LOGIDX,* or using the `FORCE_LOGIDX` server configuration keyword.

Extensive testing identified an isolated possibility of failure during automatic recovery when the *LOGIDX* logic is active. The net effect is a possibility for failed recovery in the event the c-tree Server was not properly shut down. This has been corrected as of V9.1 build 090522.

A simple solution to ensure you will not be impacted by this default setting is to disable the *LOGIDX* feature via your server configuration file. Follow these easy steps to perform this change:

1. Cleanly shut down your c-tree Server. (For example, use the c-tree Server Administrator utility, **ctadmn**, or the c-tree Server stop utility, **ctstop**.)
2. Add or change the following server configuration keyword in your server configuration file, *ctsrvr.cfg*:

   ```
   FORCE_LOGIDX   OFF
   ```

3. Restart your c-treeACE per your normal operating procedures.

2.2 Correct Handling of Segmented Files During Automatic Recovery

During automatic recovery of a failed c-treeACE process, existing file segments were renamed and thus subsequently not available to the application resulting in apparent missing data. During recovery, an attempt is made to open all segments of a segmented file found in the transaction logs. To open the segments, the segment definition resource must be read from the primary segment. However, an internal file map attribute had not been initialized resulting in a failed call. This failed call caused the recovery process to believe the additional segments did not exist. When later attempting to create the file segment, and that segment already existed, c-treeACE

then renamed the segment. The uninitialized attributes are now properly assigned avoiding this potential incorrect renaming of segmented files.

2.3 Prevent Server Termination of c-treeACE from Data and/or Index LRU Cache Miss Limitations

When a c-treeACE Server is running with the configuration options `DATA_LRU_LISTS` or `INDEX_LRU_LISTS` set greater than 1, after a large number (2^32, over 2 billion) data or index cache misses occur, the c-tree Server terminates with an unhandled exception. These options default to 4 in c-treeACE Version 9. Thus all c-treeACE Version 9 servers are susceptible to this situation. You can verify these options in the server startup information found in the server status log file, *CTSTATUS.FCS*.

When a cache page is required to hold a page that is not already in cache and multiple data or index LRU lists are in use, the data and index cache logic increments a variable used to calculate which data or index LRU list is to be used . However, the variable was declared as a signed long integer, and after 2^32 increments, became negative, resulting in a negative index offset for an array of data/index cache LRU list mutexes. Redeclaring the variables as unsigned long integers resolves this issue.

2.4 Resolved 2GB c-tree File Limitations on Linux

It was found that c-treeACE failed to read or write a c-tree file at an offset greater than 2 GB on Linux systems. An internal c-tree disk I/O function had been previously modified on Unix systems to natively support 64-bit systems. Adding the proper compiler flags when building on Linux kernels 2.4 and later resolves this 2GB file size limitation.

Note: This 2GB file limitation affected only c-treeACE 32-bit Linux Servers.

3. Fresh Additions

c-treeACE V9.1 has several significant new features that will be of interest to a wide array of developers and deployed applications.

3.1 Core c-treeACE SQL Performance Enhancements

Increased Query Performance

- Several queries involving JOINS much improved
- Faster queries through new distinct key counts
- Continual improvements to query optimization

Decreased Memory Usage

- Reduced usage in communications buffers
- Reduced usage in fetch ahead buffers
- More efficient use of internal arrays
- Minor memory Leaks in ODBC Drivers repaired

Improved Stability

- Assorted PANIC and unhandled conditions resolved
- Stabilized viewing of execution plans
- Resolved all reported unexpected server terminations

3.2 Distinct Key Count for Improved c-treeACE SQL Query Efficiency

The c-treeACE SQL query optimizer now takes advantage of distinct key counts in an index for improved efficiencies. In some cases, speedups of over 200 fold for selected queries have been reported with this additional index level information. c-treeACE SQL now creates indexes with distinct key count support by default.

3.3 Dynamic Dump Options For Enhanced Performance

When a dynamic dump runs, the disk read and write operations of the backup process can slow the performance of normal database operations. c-treeACE now supports an option that allows an administrator to reduce the performance impact of a dynamic dump.

The c-treeACE configuration option:

- `DYNAMIC_DUMP_DEFER <milliseconds>`

sets a time in milliseconds that the dynamic dump thread will sleep after each write of a 64KB block of data to the dump backup file.

An application developer can also use the c-tree **ctSETCFG()** API function to set the *DYNAMIC_DUMP_DEFER* value. For example, the following call specifies a 10-millisecond DYNAMIC_DUMP_DEFER time:

- **ctSETCFG(** *setcfgDYNAMIC_DUMP_DEFER, "10"* **);**

The *DYNAMIC_DUMP_DEFER* value set by a call to **ctSETCFG()** takes effect immediately, so this API call can be used by administrators to adjust the speed of a running dynamic dump depending on the amount of other database activity.

Note: The maximum allowed *DYNAMIC_DUMP_DEFER* time is 5000 milliseconds, set at compile-time. If a value is specified that exceeds this limit, the *DYNAMIC_DUMP_DEFER* time is set to *DYNAMIC_DUMP_DEFER_MAX*.

The c-treeACE Administrator utility, **ctadmn**, was also updated to support the dump sleep time option to change this value at run time. The "Change Server Settings" menu is available from the main menu of the **ctadmn** utility and this menu supports all of the following options:

- **Configure function monitor**
- **Configure checkpoint monitor**
- **Configure memory monitor**
- **Configure request time monitor**
- **Change dynamic dump sleep time**

3.4 Efficient Transaction Log Template Copies

The log template feature is a fast and efficient means of creating transaction logs in high volume systems. An initial transaction log template is created and copied when a new transaction log is required.

The original implementation used an operating system file copy command (for example., cp *L0000002.FCT L0000002.FCS*) to initiate the copy of the template to the newly named file. This approach required the full contents of the template file to be read. For systems experiencing high volume transaction loads where a template is frequently copied, this method placed unnecessary demand on system resources.

An improved efficient method for copying transaction log template file is available.

Template Copy Options

The following configuration options can be used to modify the speed of copying a log template such that log template disk write performance impact is reduced.

Copy Sleep Time

```
LOG_TEMPLATE_COPY_SLEEP_TIME <milliseconds>
```

This keyword results in the copying of the log template to be paused for the specified number of milliseconds each time it has written the percentage of data specified by the `LOG_TEMPLATE_COPY_SLEEP_PCT` option to the target transaction log file.

- Default value: 0 (disabled)
- Minimum value: 1
- Maximum value: 1000 (1 second sleep)

Copy Sleep Percentage

```
LOG_TEMPLATE_COPY_SLEEP_PCT <percent>
```

This keyword specifies the percentage of data that is written to the target transaction log file after which the copy operation sleeps for the number of milliseconds specified for the `LOG_TEMPLATE_COPY_SLEEP_TIME` option.

- Default value: 15
- Minimum value: 1
- Maximum value: 99

Example

The following example demonstrates the options that cause the copying of the log template file to sleep for 5 milliseconds after every 20% of the transaction log template file has been copied:

```
LOG_TEMPLATE_COPY_SLEEP_TIME 5
LOG_TEMPLATE_COPY_SLEEP_PCT 20
```

Note: If an error occurs using this method an error message is output to *CTSTATUS.FCS* (identified with the "LOG_TEMPLATE_COPY: ..." prefix) and c-treeACE then attempts the log template system copy method.

3.5 CPU Affinity Check for IBM AIX Operating Systems

c-treeACE now performs a CPU affinity check for IBM AIX operating systems. While c-treeACE can now detect the number of CPUs in an AIX system, the server configuration option, `CPU_AFFINITY`, used to set the CPU affinity for the c-treeACE process, has not been implemented for this operating system at this time. Should you need to configure c-treeACE to a particular AIX CPU resource, please consult your local system administrator for options on restricting processes to specific CPU resources.

3.6 Improved File Compaction Behavior for 6-Byte Transaction Enabled Files

Introduction

Prior to V9.1, the compact IFIL and rebuild IFIL API functions (**CMPIFILX()** and **RBLDIFILX()**) did not make assumptions about 6-byte transaction numbers for indexes associated with 6-byte transaction enabled files. As a result, regardless of if the data file was 6-byte transaction number enabled, when new indexes were recreated they lost the *ct6BTRAN* attribute. This caused errors when the server configuration specified 6-byte transaction enabled files only (for example, error **R6BT_ERR**, 745). It could also lead to transaction number overflow errors (**OTRN_ERR**, 534) on high volume systems when 4-byte transaction numbers quickly become exhausted.

When the underlying data file contains an extended header, it is more appropriate for the extended rebuild and compact functions to use 6-byte transaction numbers for the associated indices by default. As a result, any data file containing an extended header that is compacted or rebuilt, will now have have the 6-byte transaction number attribute enabled by default.

New CMPIFILX8 API

The **CMPIFILX()** extended API call does not facilitate an array of *XCREblk*s to be passed in. When the indices associated with the data file to be compacted exist, then compact generally does not need these *XCREblk*'s used when the indices were created. However, if one or more of the indices does not exist, then **CMPIFILX()** recreated a *XCREblk* from the data file. But this does not account for differences between the original data file *XCREblk* and the original index file *XCREblk*.

An extended header version of **CMPIFILX()** has been introduced as a step to alleviate this situation.

This new API call declaration is:

```
ctCONV COUNT ctDECL CMPIFILX8(pIFIL ifilptr, pTEXT dataextn, pTEXT indxextn,LONG permmask, pTEXT groupid,
pTEXT fileword, pXCREblk pxcreblk)
```

Secondly, when either **CMPIFILX()** or **CMPIFILX8()** are run on the server (as opposed to stand-alone) and the server configuration option `COMPATIBILITY EXTENDED_TRAN_ONLY` is specified, then the *ct6BTRAN* mode bit is turned on in the recreated *XCREblk* or the explicit *XCREblk*, respectively, to ensure 6-byte transaction number support.

It should be noted that since **CMPIFILX()** reconstructs an *XCREblk* for the index from the data file, there may be attributes of the data file that were not part of the original index file definition. For example, if the data file supported a large extent size, then the index inherits this large extent size regardless of how it was originally created.

If the configuration option, `COMPATIBILITY 6BTRAN_NOT_DEFAULT`, is specified, the *XCREblk* array passed into **RBLIFILX8()** and **CMPIFILX8()** can still explicitly set the *ct6BTRAN* attribute bit. Likewise, if *ctNO6BTRAN* is passed in explicitly in calls to **CMPIFILX8()** and **RBLIFILX8()** then the default behavior is overridden, and *ct6BTRAN* will not be turned on.

Compact File Utility ctcmpcif

`#define USEXCREBLK` and `#define LOCALGETXCREBLK` have been added to the compact *IFIL* utility source code (*cmpifil.c*, as in *ctrbldif.c,*) to control definition of the data and index *XCREblks* using the new **CMPIFILX8()** API call. This option is off by default. Uncomment this #define should you desire this functionality in the compact *IFIL* utility.

3.7 Added Platforms and Environments

- AIX 6.0
- Mac OS X 10.6 (Snow Leopard)
- Windows 7
- CodeGear 2009 (Including VCL and c-treeACE SQL dbX Components)
- Sun Studio 12
- c-treeACE SQL for QNX

4. Notable Resolved Issues

c-treeACE V9.1 contains minor modifications that address items found in continued testing as well as several reported issues since the V9 release.

4.1 Windows Resource Error (1450) Configurable Retry Logic

When the Windows kernel has allocated all of its paged-pool memory, it will not be able to perform many tasks and instead returns a STATUS_INSUFFICIENT_RESOURCES (0xC000009A) message. This is a restriction of 32-bit addressing (only 2GB addressable within the kernel), regardless of the amount of memory available in the system.

When the FairCom Server configuration option `IO_ERROR_BLOCK_SIZE` option is specified in the FairCom Server configuration file, a read or write operation that fails with Windows system error 1450 (ERROR_NO_SYSTEM_RESOURCES) is retried in blocks of the specified size. If any one of those read or write operations fails, the FairCom Server fails the read or write operation.

The FairCom Server supports two additional configuration options that permit additional disk read/write retries and a sleep interval between retries.

`IO_ERROR_BLOCK_RETRY <retries>` specifies the maximum number of failed IO_ERROR_BLOCK_SIZE-sized I/O operations that must occur before the I/O operation is considered to have failed. If the IO_ERROR_BLOCK_SIZE-sized I/O operations that are being attempted for a particular I/O operation fail more than *<retries>* times, the FairCom Server writes a **READ_ERR** (36) or **WRITE_ERR** (37) message to *CTSTATUS.FCS* and considers the I/O operation to have failed.

A value of -1 signifies infinite retries. The default is 0, which means that the I/O operation is tried only once in IO_ERROR_BLOCK_SIZE-sized blocks, and if any of these I/O operations fails, the entire I/O operation is considered to have failed. As another example, if IO_ERROR_BLOCK_RETRY is set to 20 and IO_ERROR_BLOCK_SIZE is set to 65536, if a 327680-byte write is retried as 5 65536-byte write operations, then the I/O operation fails if there are 20 failures to perform those 5 write operations.

`IO_ERROR_BLOCK_SLEEP <time>` specifies a time in milliseconds between retry attempts. The default is zero, which means that retries are attempted immediately.

SNAPSHOT Monitoring of Failed Retires

To permit monitoring the number of I/O error 1450 retries that have occurred, a counter has been added to the system snapshot structure. The *sctioblkretry* field of the *ctGSMS* structure is defined as an unsigned long integer that stores the total number of I/O error 1450 retries that have occurred since the FairCom Server started. The snapshot log file *SNAPSHOT.FCS* displays the I/O error 1450 retry counter value with a description of "I/O ERR(1450) automatic retries:". The

system snapshot structure version has been changed from 9 to 10 to note the presence of this new field in the structure and the statistics monitoring utility, **ctstat**, and **ctsnpr** utilities have been updated to properly handle the presence of this field in the system snapshot structure and snapshot log.

4.2 Improved Thread Safety of System Time Calls

A review of **localtime()** behavior has determined this call was not thread safe on all operating systems. Use of this function, and related functions, has been modified. In the case of **localtime()**, **localtime_r()** is now used, and similarly with other functions included from the system time libraries.

Note: This system call was also included in the default SQL Types SDK custom callback libraries found in the *ctsqlcbk.c* module utilized by some customers. This module has been modified as well when available in our code repositories.

4.3 Correct Conditional Index Results with Zero-Length String Literals

An index specified the conditional index expression:

```
!strnicmp(myField, "", 6)
```

and the data file contains records having values of *myField* that match this condition (that is, *myField* is ""). However, for c-treeACE V9, the expression did not evaluate as true for these field values. The same conditional expression produced the expected results in c-tree Plus V8. The V9 conditional expression logic parses zero-length (empty) string literals differently than V8. A NULL pointer was placed into the expression tree rather than allocating space to hold an empty string and adding that value into the expression tree. The string comparison function returns a non-zero value when only one of its parameters (*Source* or *Dest*) is NULL, and caused the expression to return unexpected results in this situation.

The function that adds a string literal value to the expression tree was modified to allocate a one-byte buffer when the string length is zero. Similar logic was also reviewed for other cases where an empty string literal value is not properly handled and these other functions were modified accordingly.

4.4 Allow c-treeACE V9 Conditional Expression Parser Field Names to Match Data Type Names

It was found that c-treeACE V9 failed to parse some conditional expressions that were allowed in the c-tree Plus V8 SDK. In V9, the expression parser for conditional indexes has been moved into the FairCom DB API layer and that logic includes support for a *CAST* function:

```
 F_CAST '(' Expression F_AS ATYPE ')'
 F_CAST '(' Expression ',' ATYPE ')'
```

ATYPE is a data type which can be any of the type names from the *symtab* list defined in *ctdbcrun.c*. This new logic could cause some prior expressions to fail. For example, one data type is named *NUMBER* and when used in an expression such as:

```
strlen(Number) > 0
```

c-treeACE V9 considers "Number" as a data type and not as a field name, and fails to parse the expression.

To support previous expressions, the grammar rules for the cast function have been modified to use the symbolic *IDENTIFIER* instead of *ATYPE*. Now, in the rules for the cast function, after finding an *IDENTIFIER*, a check is made if the *IDENTIFIER* is the name of a data type. If not, an error is returned. Otherwise, the data type is added to the parse tree and expression parsing continues.

4.5 Improved Performance of Delete Node Queue Management

A very long server shutdown was noticed in a particular environment due to delete node processing. Two areas of server operation were enhanced to minimize this time spent in delete node processing.

Delete Node Queue Entries

Previously, the delete node queue exercised the following pseudo code for each queue entry:

- read queue
- open file
- begin transaction (for *TRANPROC* files)
- prune tree
- commit transaction
- close file

This basic loop was modified in two ways:

1. An attempt is made to peek at the next queue entry to determine if the next tree pruning will be for the same index file (the host index or one of its members).
2. If the files match, skip the commit/begin operations and skip the close/open operations. More precisely, the number of skipped transaction commits and skipped file closures are actually tracked and an artificially imposed limit is set on the number of skips to avoid very long transactions and keeping the file open in shared mode for an extensive period.

By default, *NODEQ_TRNLEN* is set to ten (10), and *NODEQ_OPNLEN* is set to twenty (20). Once shutdown begins, these limits are increased by a factor of ten.

Limited testing shows that shutdown processing of the delete queue runs at least twice as fast with this new behavior.

Note: These changes have little effect if the queue entries are widely varied across different files.

c-tree Server Shutdown Processing

The c-tree Server now supports a configurable limit on the number of delete node queue entries that the c-tree Server processes when it is shutting down. If the option `DNODEQ_SHUTDOWN_LIMIT` is specified in the c-tree Server configuration file, then when the c-tree Server shuts down, if there are more than the specified number of entries in the delete node queue, the delete node thread writes all the unique queue entries to a disk stream file named *DNODEQUE.FCS*. A memory-based index file is used to eliminate duplicate queue entries, and only the unique entries are written to disk. If the number of unique entries is less than `DNODEQ_SHUTDOWN_LIMIT`, those entries are returned to the delete node queue and are processed by the delete node thread before the c-tree Server completes the shut down.

`DNODEQ_SHUTDOWN_LIMIT 0` causes the c-tree Server to process all entries in the delete node queue when shutting down.

The c-tree Server always attempts to open and read all entries from the file *DNODEQUE.FCS* into the delete node queue at startup, regardless of the `DNODEQ_SHUTDOWN_LIMIT` setting. The c-tree Server deletes the file *DNODEQUE.FCS* after populating the delete node queue with its contents. An administrator can delete the file *DNODEQUE.FCS* before starting the c-tree Server to avoid processing these persistent delete node queue entries.

4.6 PreImage Memory Files are no Longer Promoted to TRANLOG files During a Dynamic Dump

The c-treeACE `PREIMAGE_DUMP YES` configuration option promotes *PREIMG* files to *TRANLOG* files during a dynamic dump. An unintended side effect of this option was that *PREIMG* memory files were also promoted to *TRANLOG* files. As a result, memory index nodes could be placed onto the list of updated buffers for transaction controlled files. As these buffers are never flushed to disk (because memory files are never written to disk) these entries resulted in the number of active transaction logs to increase, as c-treeACE keeps all transaction logs beginning with the transaction log in which the transaction on the memory file promoted to the *TRANLOG* mode occurred.

When a dynamic dump is performed with the `PREIMAGE_DUMP YES` configuration option specified in *ctsrvr.cfg*, and *PREIMG* memory files are open, c-treeACE could exhibit any of the following symptoms:

- The number of transaction logs may increase, with *CTSTATUS.FCS* showing a reason of "Cache/Buffer Pending Flush"
- Creating a memory data file during a dynamic dump fails with error **ITIM_ERR** (160)
- c-treeACE may terminate during a checkpoint
- Automatic recovery fails with error 12 attempting to open a non-existent memory data file

Dynamic dump logic has been modified such that *PREIMG* memory files are no longer promoted to *TRANLOG* files to prevent this situation.

4.7 LOG_ENCRYPT Option Now Correctly Supported with Advanced Encryption

c-treeACE failed to start with a log incompatibility error **LFRM_ERR** (666) when both the log encryption option `LOG_ENCRYPT` and advanced encryption (`ADVANCED_ENCRYPTION YES`) options were in effect. c-treeACE would even fail to start in a directory with no existing transaction log files.

Comprehensive checks during c-treeACE startup are now done to ensure that log encryption settings are appropriate for the type of encryption for existing transaction logs. When an incompatibility is found, c-treeACE startup fails with an error message in *CTSTATUS.FCS* indicating which options must be changed to access the transaction log files.

The table below shows the expected results for the possible combinations of Log Encryption and Advanced Encryption for each possible transaction log encryption type used by an existing log file.

Log Encryption Option	**Advanced Encryption Option**	**Log Encryption Type**	**Expected result**
N	N	None	OK
N	Y	None	OK
Y	N	None	OK, new logs are scrambled with *CAMO** technology
Y	Y	None	OK, new logs encrypted with advanced encryption
N	N	*CAMO**	Error: Enable log encryption to proceed
N	Y	*CAMO**	Error: Enable log encryption and disable advanced encryption to proceed
Y	N	*CAMO**	OK
Y	Y	*CAMO**	Error: Disable advanced encryption to proceed
N	N	Advanced	Error: Enable log encryption and advanced encryption to proceed
N	Y	Advanced	Error: Enable log encryption to proceed
Y	N	Advanced	Error: Enable advanced encryption to proceed
Y	Y	Advanced	OK

* CAMO or "Camouflage" is an older, legacy method of hiding data, which is not a standards-conforming encryption scheme, such as AES. It is not intended as a replacement for Advanced Encryption or other security systems.

5. Features and Fixes

5.1 Resolved c-treeSQL Panic with WHERE Clause Subqueries

A c-treeSQL query involving an ON clause with a WHERE clause containing a subquery was found to generate a panic condition with the following message:

```
TPEUTIL et_pexpr::psr_chg_penum penode not found 2
```

The query optimizer was found to incorrectly build an internal list in the processing of a query tree of this nature and has been modified to avoid this panic condition.

5.2 Prevent c-treeACE SQL crash with OR-ed subquery

Perceptive Software has reported a series of crashes at customer production sites recently. Upon detailed analysis of several of the mini dump logs obtained from the customers, and through manually forcing the c-treeSQL Server into a similar state as that observed in the mini dump logs, the FairCom R&D team has isolated the crash to the following query:

```
SELECT DISTINCT TT.* FROM IN_TASK_TEMPLATE TT INNER JOIN IN_USR_TASK_TEMPLATE_PRIV TTP ON TTP.USR_ID IN
( SELECT GROUP_USR_ID FROM IN_USR_GROUP WHERE USR_ID = ? ) OR TTP.USR_ID = ? WHERE TTP.PRIV_ID IN ( 15000
) AND TT.TASK_TEMPLATE_ID = TTP.TASK_TEMPLATE_ID AND TT.IS_ACTIVE = 1 ORDER BY TT.TASK_TEMPLATE_NAME;
```

The appropriate internal fix to the c-treeSQL Server has been applied to this line of c-treeACE SQL.

5.3 Resolved Unhandled Exceptions When Executing an Explain Plan with c-treeACE

When an explain plan operation is executed on a cached statement, c-treeACE terminated with an unhandled exception. An object had been freed, however, a pointer to that object had not been reset. This pointer value is now correctly set to NULL when the referenced object is freed.

5.4 Corrected c-treeACE SQL Panic Condition Relating to Internal Dictionary Caches

A c-treeACE SQL PANIC was observed referencing the following condition:

```
ddm_get_rssid 2.
```

It was found that internal dictionary information that was not yet committed could be placed into the working caches. New logic has been put into place to ensure that only fully committed changes are inserted into these critical working caches.

5.5 Corrected c-treeACE SQL Memory Exception when Copying Databases

When the c-treeACE SQL database utility, **ctsqlcdb**, was used to create a copy of a c-treeACE SQL database, the server process could crash. The source of the crash was determined to be an improper memory free operation. The specific parameters to the internal routines to allocate and free this memory resource were modified to avoid this situation.

5.6 Resolved c-treeACE SQL Server Crash on Unix Systems

c-treeACE SQL on the Solaris operating system was found to not protect a memory copy from the communication buffer from a memory overwrite condition, and a crash was reported as a result of this failure. The windows version of the server already included this protection, and the same logic was applied to the Unix versions of the server.

5.7 Improved Join Query Performance

A c-treeACE SQL query was observed to execute much slower after a recent code revision. Changes to enhance the selectivity for operators other than EQ had impacted this performance. A new approach to the selectivity calculation, as well as the case when the index is unique, has been taken which alleviates this slow query time.

5.8 c-treeACE SQL Query Performance Improvements

Several c-treeACE SQL queries were reported to have poor performance. Performance was improved by including outer joins in join order optimization. (whereas before, the optimizer was mixing INNER and OUTER joins.) Additional improvements were made in NL join optimization such that if there is only one tuple coming from the left branch there is no need to introduce a 'Project into temp' node on the right branch.

Analysis of these queries also revealed common areas that could be improved within c-treeACE SQL data handling functions. These include the following additional modifications:

- Avoid unnecessary memory allocations and releases for temporary data copies.
- Reduce the number of calls needed when retrieving a field from a record.
- Avoid unnecessary context switch calls that result in no operation.
- Reduce the frequency of checking user and transaction states. These checks are performed for features such as query timeouts and killed users. Abandoned transactions due to dynamic dumps or a server quiesce can also lead to invalid transaction states that must be checked. These checks have been reduced to no more than once per second.

5.9 c-treeACE SQL Query Performance Improvements

A number of c-treeACE SQL queries were reported to return invalid results, or had performed poorly in testing. The c-treeACE SQL engine was modified with several query optimizations to address these specific cases.

- **Missing rows from an outer join.** An outer join is converted to an inner join if there is an equality join predicate in an inner join node above any of the right hand side columns of the outer join. An error was found and corrected in forming the execution plan in determining whether a predicate was from the right hand side of the outer join.
- **Improved Read Performance.** By adjusting the underlying ISAM storage engine key buffering logic, better performance is now obtained with c-treeACE SQL as this avoids updating all ISAM key buffers during a read operation.
- **Slow performing Queries.** Several reported slow queries have been improved with various optimization enhancements.

5.10 Improved Query Performance with Optimized Join Orders

A slow query was reported where the join strategy picked was not the best one based on inspection of the execution plan. The join order optimization logic has been adjusted so that it recognizes table pairs t1, t2 such that:

1. t1 has no index on the join column;
2. t2 has a unique index on the join column;
3. t2 does not have any other join condition with any other table;
4. The join condition between t1 and t2 is on a single field.

After this the remaining tables, excluding table t2, are ordered. After the join order is obtained t1 is replaced with [t1 join t2] , t2 being to the right of t1. Note that the cardinality of [t1 join t2] will be the same as that of t1, as t2 has an unique index on the join column.

5.11 Faster Queries Through a Distinct Key Count

It was found that a complex (not prepared) query could run slow with given search values compared with other values. The query consisted of many inner and outer joins on large tables with duplicate indices containing few keys matching the vast majority of the records and other keys matching very few records. To better optimize these queries required an accurate estimation of the duplicate index selectivity, which implies knowing how many distinct keys there are in an index allowing duplicates. This feature has been added for c-tree indices.

To support this feature, an index requires a distinct key count attribute. A utility, **ctdbdistinct**, is available to add the distinct key count feature to existing indices in a c-treeSQL database.

The c-treeSQL Server query optimizer was then updated to take advantage of this distinct key count. With this feature enabled, all indices are created by default with this attribute available.

5.12 c-treeACE SQL ON Clause Join Order Optimizations

It was possible, while the c-treeACE SQL engine was forming the join order, the sorting requirement was not considered. The optimizer has been modified to take into consideration any sorting requirements during the join order optimization phase.

Specifically, predicates in the ON clause, were not pushed down the right branch of the outer join node, however, predicates from the WHERE clause were pushed down to the right branch of a Left Outer Join (LOJ) and the LOJ converted to an inner join. The modifications now allow pushing down of predicates from the ON clause to the right branch of the LOJ, however, without converting the LOJ to Inner Join.

5.13 Improved JOIN Optimizations when Predicates are Included

A slow query was reported that consumed a large percentage of CPU time. The query contained a JOIN operation with predicates and a where clause with OR-d conditions. It was identified that OR optimization was not taking place if there existed join predicates in a join node. In the case of a JOIN syntax where an ON clause is specified, join predicates always exist. Hence, the condition that checked for the existence of join predicates that disabled OR optimization was removed.

5.14 Correct c-treeACE SQL EXTRACT Scalar Function Ranges

A query such as the following returned an unexpected overflow error:

```
SELECT EXTRACT(DAY from '03-31-2009') from syscalctable;
```

The scalar function EXTRACT failed to properly handle correct day ranges for all dates. The range checks have been modified to correctly process all valid date ranges.

5.15 Corrected TOP ... ORDER BY ... c-treeSQL Query Results

A query of the form `TOP .... ORDER BY ...` produced an incorrect result set.

An optimization to the TOP N clause introduced this faulty logic. Improved maintenance of the cached blocks and storing direct references (as opposed to indirect pointers) to the target tuples resolves this error.

5.16 Correct LEFT OUTER JOINS Containing OR Conjunctions in an ON Clause

A query containing an OR conjunction in the ON clause of a left outer join returned incorrect results. Optimizer logic was found to inappropriately apply restrictions to this statement resulting in the incorrect return. This logic as adjusted to properly handle this clause in the case of a left outer join.

5.17 Correct c-treeACE SQL Query Returns With BETWEEN Conditions

A query containing 3 conditions on the same field using > and < operator was found to return the wrong result. When possible, the c-treeACE SQL engine substitutes two predicates on the same column with a between predicate. After creating the between predicate, the predicates that have been replace by BETWEEN are removed as they are now redundant, however, other predicates on the same column were also incorrectly removed. c-treeACE SQL query logic was adjusted such that if the index operator is BETWEEN, then only those predicates that were used to form the BETWEEN predicate are considered to be redundant.

5.18 Correct String Comparisons Within Case Insensitive Databases

A query of the system tables of a database that was created case insensitive (which is not the default), having a condition on a field not indexed, failed to retrieve the proper rows. Internal logic uses c-tree expressions that can perform comparison on strings using the "regular" comparison operators (=, >, < ...). However, these operators on strings are case sensitive. In the case of a case insensitive database this same logic cannot be used. New logic has been added such that case insensitive databases are identified as such and when comparing text fields within these databases, a proper string comparison is performed.

5.19 Prevent Re-use of Prepared c-treeACE SQL Statement Execution Errors

A prepared c-treeACE SQL statement could return an unexpected error from an earlier execution. For example, a query such as:

```
SELECT * FROM test WHERE ( f1 =  ? )
```

using a parameter value "text" where f1 is an integer data type results in the error "invalid integer value". The error code was stored in an internal cache and subsequent executions with valid parameter values returned this same error. A check is now done to ensure the cache is properly validated before such an error return.

5.20 Resolved DADV_ERR After c-treeSQL Transaction Isolation Level Change

A **DADV_ERR** (57, Proper lock is not held) was observed after reducing the transaction isolation level below a repeatable read isolation level. An internal flag maintains the state of the isolation levels for any particular thread. Previously, this flag was only reset when a strict serialization transaction isolation level was in involved. The behavior of updating this flag has been changed such that it is now reset upon any change of isolation level avoiding this lock error condition.

5.21 Correct Handling of c-treeSQL CONTAINS Clauses Larger than 127 Bytes

A c-treeSQL CONTAINS clause consisting of more than 127 bytes could potentially crash the c-treeSQL Server. An internal array declared as a signed char was used to accommodate more than 128 bytes. Changing this declaration to an appropriately sized unsigned value avoids any illegal array access.

5.22 Corrected Error 17101 During a DELETE Statement

An error -17101 while running a c-treeACE SQL delete statement was observed. An optimization of the sort node was found to not be handled correctly. A new project node just above the restrict node is now included in the execution plan to project only the required project expressions that prevents this error.

5.23 Corrected Error 17466 on Index Creation

An error 17466 (CT - Incremental Index - dfilno not a ISAM file) was observed when adding an index in c-treeACE SQL. During a pending delete an internal file control block member could be

found in an unexpected state when re-used, as in this case, by an index creation. This internal flag is now explicitly set for any pending delete to avoid this error.

5.24 Avoid Unnecessary E69 Errors in CTSTATUS.FCS

Error **BNOD_ERR** (69) was found in the *CTSTATUS.FCS* server log file, however, no error had been returned to the client. Additionally, the index appeared to be intact. A test case that logged **BNOD_ERR** to *CTSTATUS.FCS* found the c-tree function called actually succeeded. Logic has been added to suppress the **BNOD_ERR** message to *CTSTATUS.FCS* when the error is properly handled internally.

5.25 Correct Propagation of Errors in c-treeSQL

An function call used by the c-treeSQL Server failed with error **KDEL_ERR** (4), however, this error was not properly propagated to the application that launched the calling SQL statement. An incorrect return value of STATUS_OK was always returned regardless of any actual error that may have been encountered. The correct status variable is now properly returned in all cases.

5.26 Avoid c-treeACE JVM Error Messages when the JVM Environment is not Present

To avoid concern from end users, it was requested that the JVM Error messages could be avoided in the case that c-treeACE was started without a JVM environment present. A server configuration keyword is now available to suppress these messages.

```
SQL_OPTION SUPPRESS_JVM_LOAD_ERR
```

When this configuration keyword is specified the only message on the console in case of Java environment errors will be "No JAVA capabilities" at server startup.

5.27 Allow Deletion of Self Referential Foreign Keys in c-treeACE SQL

It is possible to have a Foreign Key reference its own table. However, it was noted that with c-treeACE SQL, if a record is referenced by itself, it could not be deleted. Other SQL databases allow this action.

In this situation, the delete action was found to violate constraints, due to the order of constraint checking within the c-treeACE SQL engine, and c-treeACE SQL did not take into account it was referencing the same record.

The constraint check logic was adjusted such that the check is made after the actual deletion, avoiding the constraint violation.

5.28 Configurable c-treeACE SQL PRESERVE CURSOR Option

An end user received the error 20125, "Number of open cursors exceeds limit", while running Crystal Reports with the c-treeACE SQL ODBC Driver. Perceptive's specific c-treeACE SQL ODBC driver always enabled PRESERVE CURSOR, regardless of what was set in the ODBC administrator. The PRESERVE CURSOR behavior is now a configurable option from within the ODBC administrator.

Note: The direct link ODBC driver still defaults to PRESERVE CURSOR behavior.

5.29 SNAPSHOT Statistics for c-treeACE SQL Statement Cache Counters

The *SNAPSHOT* API can now report on c-treeACE SQL statement cache counters. The following information was added to each *gcache_t* object.

```
struct gcache_snapshot
{
   dh_longlong_t     requests;
   dh_longlong_t     hits;
   dh_i32_t          max_entries;
   dh_i32_t          entries;
   dh_i32_t          high_entries; /* highest number on entries */
};
```

This information is available through the **ctSNAPSHOT()** call for the static and dynamic statement cache objects. At this time this feature is available only on the Windows platform.

5.30 Updated Version Information in c-treeACE SQL Installation

The *ctreedbs.dll* module was found to not be upgraded in some c-treeACE SQL installations due to unchanged version information. The version information structure in this module is now updated to reflect new versions allowing it to be detected and upgraded during an installation.

5.31 Built-in Stored Procedure to Switch Transaction Mode

It can be useful for performance reasons to avoid transaction memory usage in some common maintenance operations. Consider the case of upgrading a large database where some tables are added with a SQL query from another table that returns a large number of rows. As this occurs inside a single SQL statement the transaction consumes memory until committed. For large files, this can potentially exhaust memory on some systems and result in complete failure.

c-treeACE can allow the transaction processing mode of a file to be disabled which avoids consuming memory for a very large operation. To make this available to c-treeACE SQL, a built-in stored procedure allowing the transaction mode of a file to be changed has been added. This operation requires exclusive file access.

```
fc_set_file_tran_state(VARCHAR owner, VARCHAR tablename, TINYINT mode)
```

Valid values for *mode* are

- 0 : No transaction control.
- 1 : Transaction control without recoverability. (*ctPREIMG*)
- 2 : Transaction control and recoverability. (*ctTRNLOG*)

When a c-treeACE SQL file is set to mode of 0 (no transaction control), all changes to the file will have immediate and permanent effect, such that a ROLLBACK operation will have no effect on this file. If another user is able to open the file, isolation levels are undefined. Furthermore, c-treeACE SQL operations that modify multiple rows of this table will not be atomic in the event of an error.

For update statements, it is important to ensure no errors are encountered during execution. For example, a statement such as

```
UPDATE tbl SET col=99
```

that encounters an error midway through the update, will leave the state of the table undefined and the operation should be re-examined. It is possible it will be necessary to restore from a backup to return to a previous state.

Because of these effects, the intended use of this mode is limited in scope. No file definition/schema changes to this file should occur while in this mode. All previous schema changes should be committed before calling this procedure. If an error occurs while operating on a file with a mode 0, the table should be deleted and the transaction rolled back.

Note: The stored procedure **fc_check_file_tran_state()** can be used to ensure that no files remain in modes other than the database default.

5.32 Improved Compile Options for the c-treeACE SQL Direct Link ODBC Driver

The c-treeACE SQL direct link ODBC driver was found to periodically crash an application when connecting to the server. A compile time option (*-Bsymbolic*) was modified to ensure the correct C++ **new()** operator is resolved rather than an overloaded method used by the driver due to library load order.

5.33 Signal Batch Clean-up on Client Side

It was found that redundant calls to **BATSETX()**/**BATSET()** with a mode of *BAT_CAN* could be eliminated for convenience and notify a client when a batch request has been completed.

This feature causes *sysiocod* to be set to **BTNO_COD** (-844) on calls to **BATSETX()**/**BATSET()** that result in the current batch being closed. If the client application finds *sysiocod* set to

BTNO_COD, then the application does not need to issue a *BAT_CAN* call before issuing a new batch request.

This new behavior requires the following FairCom Server configuration entry:

```
COMPATIBILITY   BATCH_SIGNAL
```

5.34 Automatic Batch Close Mode

Calls to **BATSETX()**/**BATSET()** that "exhaust" the batch do not typically close (or clean-up) the batch. The next call will return a **BTMT_ERR** (428) indicating the batch has no more remaining entries, and the batch is closed. More specifically, this may occur when *BAT_GET* or *BAT_NXT* requests are made for: partial key requests (*BAT_PKEY*); greater or less than key requests (*BAT_GKEY*, *BAT_LKEY*); and physical order requests (*BAT_PHYS*). These "exhausted" conditions can be detected, and we now close the batch and use the new *sysiocod* signal.

An application calling **BATSETX()** can now OR *BAT_CLSE* to the *mode* parameter. Then when a *BAT_PKEY*, *BAT_RNGE*, *BAT_GKEY*, *BAT_LKEY* or *BAT_PHYS* request exhausts all the entries, either in the first call to **BATSETX()** or in subsequent calls, the batch will be closed immediately, and *sysiocod* will be set to **BTNO_COD** to allow the application to know the batch has been closed.

By exhausting a batch we mean returning all the requested information (records, keys, or record positions) which satisfy the batch. Without this modification, an exhausted batch typically requires one additional call (*BAT_NXT*) to detect the batch is exhausted and cause the batch to be closed, or a call to explicitly request the batch close (*BAT_CAN*).

See also:

- *First Call*

5.35 Enhanced Batch Communication with the c-tree Server

For enhanced performance, a c-tree batch modification has been made to reduce the number of bytes sent by a client application to the server when a partial key request is made to the **BATSET()** call. Now, instead of sending the entire *PKEYREQ* structure to the server, only the fixed state variables (the 14 byte *PKEYREQ* prefix) plus the significant bytes of the target are sent to the server. Without the modification, **ctSIZE(***PKEYREQ***)** bytes were sent (including *MAXLEN* key target bytes) regardless of the number of significant bytes.

This change is client-side only. The server normally recreates a full-size *PKEYREQ* block when the request block is smaller than **ctSIZE(***PKEYREQ***)**.

5.36 Improved CTUSERX() Functionality for Custom Server Operations

CTUSERX() provides a means to enable your own custom server side functionality for a client to access. This module is part of the c-treeACE Server SDK.

The original implementation of the **CTUSERX()** client-side code did not send the first two function parameters (*ctlbufptr* and *ctlbufsiz*) to the server, and it assumed the input buffer (*inbufptr*) to be string data. **CTUSERX()** was modified to send all parameters to the server and additionally allow binary data in this input buffer.

The *ctlbufsiz* parameter was removed from the **CTUSERX()** function definition, as the control buffer data is passed to the c-tree Server as a string, and so the client code computes its string length rather than passing *ctlbufsiz* to the server.

The function prototypes for both the client **CTUSERX()** and server-side **CT_USERX()** functions now become:

```
extern ctCONV LONG ctDECL CT_USERX( pTEXT ctlbufptr, pTEXT inbufptr, VRLEN inbufsiz, pTEXT outbufptr, pVRLEN outbufsiz);

extern ctCONV LONG ctDECL CTUSERX ( pTEXT ctlbufptr, pTEXT inbufptr, VRLEN inbufsiz, pTEXT outbufptr, pVRLEN outbufsiz);
```

Further extending behavior, **CTUSERX()** was modified to support loading a **CTUSER()** dynamic library and executing a specified function, similar to the existing **CTUSER()** function loading. The function exported by the **CTUSER()** shared library must conform to the same prototype as above.

Note that the last parameter of **CTUSERX()** has been changed to a pointer to the output buffer length (type *pVRLEN*) The caller passes the address of a *VRLEN* variable whose value is the size of the output buffer. On return, the value is set to the number of bytes written by c-treeACE to the output buffer.

Note: If a client calls the new version of the **CTUSERX()** client-side function when connected to c-treeACE that does not support the new **CTUSERX()** function, the server returns error **SFUN_ERR** (170, bad function number). c-treeACE continues to support calls by clients that use the original **CTUSERX()** function.

Backward Compatibility

To restore the original client-side **CTUSERX()** behavior, add #define NO_ctBEHAV_CTUSERX to *ctoptn.h* and recompile the c-tree client library.

5.37 Prevent Potential Rebuilt Data Corruption

A **RRED_ERR** (407) occurred unexpectedly after a file rebuild or automatic recovery operation. A potential exists for *HUGE* files that are nonTRANDEP (fixed or variable length) to experience this error after these operations. At issue is the length of resource headers which are longer than an expected header limit contained in transaction log entries. Another symptom of this problem are unexpected 'ff' values contained in the resulting rebuilt data file overwriting once valid data records. While this situation should be considered a rare experience under normal operating

conditions, the potential for data loss and/or corruption exists. Therefore, this fix is considered a critical fix in the specified situations.

5.38 Improved File Block API Calls

A low-level **ctFILBLK()** call on an index file that contains more than one additional member may fail with an **FUSE_ERR** (46, file number already in use) error.

When establishing a low-level file block, the file block logic requires a user file number at the start of a consecutive range of two user file numbers that are not in use. However, an index having more than one additional member needs more than two consecutive user file numbers, and if the required number of user file numbers is not available, the attempt to open the index fails with **FUSE_ERR**. The file block logic now requests consecutive numbers up front to ensure an appropriate file number is obtained.

The **ctFILBLK()** could also hang in particular cases.The file block routine has a polling loop that attempts to clear threads out of the c-tree core. The logic first set a status byte on all threads that have the file(s) to be blocked open and are inside the core. Then it checks to see if any threads that don't respond to the status byte change are being blocked (for example., waiting for a lock to be released). If so, it attempts to "kill" the blocks. However, the logic did not account for an extreme timing issue that would permit the block to be acquired after the checks for blocks had been completed. The result was that the file block routine looped indefinitely without making progress on the object of clearing threads from the core.

Now the polling loop periodically rechecks the thread status, and it can clear blocks on subsequent passes around the loop. The thread that had its blocking lock request killed (the second application) returns from the lock request with an error **FBLK_PND** (801). This error signifies that a block attempt required the thread (application) to not complete its last request, and instead leave the core code.

In addition to fixing this specific looping problem, additional changes have been made to the logic:

1. An external request to kill the file block thread that occurs during the polling loop will now be detected, and the file block will be aborted and return a **TUSR_ERR** (7),
2. The polling is monitored such that in the event another unforeseen situation leads to the loop repeating without making any progress reducing the number of threads still in the core code, the file block routine will abort itself, returning an **FBLK_ABT** (842) error,
3. Messages are sent to *CTSTATUS.FCS* when it appears the polling loop is having difficulty clearing the threads,
4. Existing logic to recover from an error occurring after the polling loop has been corrected to ensure that the file block attempt is completely cleaned up, and this revised error recovery is also used in cases (1) and (2) above.

Note: The second thread in the above test will still receive an **FBLK_PND** (801) error when the file block routine is aborted in cases (1) and (2) above. Whenever a thread (application) receives an **FBLK_PND** (801) error it is free to retry the operation. If the block was successfully obtained, the retry will behave in accordance with the type of file block requested (the thread may be suspended or return an error). If the block was aborted (as in cases (1) and (2) above), then the retry should behave just as if no file block attempt had been made.

5.39 Disable Reuse of Deleted Space in Variable Length Transaction Controlled Memory Files

A feature introduced in V8.27 allowed more efficient reuse of space in transaction controlled variable length records. It was found, however, that an update of such a record that was a memory file record failed with an **ITIM_ERR** error (160) on a commit of the transaction. A further symptom of this error was high CPU usage by the server after this event.

When a memory record was updated such that its size was reduced, and if the unused space at the end of the record is large enough, a variable-length header (containing a delete mark) was written at the start of the unused portion of the record. For a memory file, this resulted in a 160 error when updating the memory record images at the end of the transaction, as the memory address of this change to the (now new) record image does not correspond to the start of a memory record maintained in the memory file hash table.

As reuse of space is not appropriate for memory files, the marking of space as deleted is no longer done for this c-tree file type.

Note: The space management feature can also be disabled for all transaction controlled files with the server configuration keyword: `COMPATIBILITY NO_VARLEN_TRAN_UNUSED`

5.40 Corrected LONGVARCHAR Field Handling with Fetch Ahead Logic

When marshalling parameters into the *SQLDA* object for JDBC, c-treeSQL fetch ahead logic inappropriately read a buffer that was allocated one less than required. An off by one error occurred when calculating the start of the field, resulting in the field to be marshalled as a VARCHAR field rather than a LONGVARCHAR. This mismatch generated the internal error (-20000). The length of the allocated buffer has been properly sized now avoiding this error.

5.41 Configuration of Server SQLDA Structure Size for Reduced Memory Usage with c-treeSQL

The c-treeACE SQL *SQLDA* is a storage area for descriptive information pertaining to dynamic SQL statements. An *SQLDA* can be used while passing parameter values and while retrieving results.

The size of this structure has been reduced in an effort to reduce memory usage. The default *SQLDA* structure has been reduced. This parameter controls the size of the structure in the server and the size of the fetch ahead buffer.

This size can be configured with the c-treeACE SQL configuration keyword:

```
SETENV DH_SVR_DA_BUFFER
```

In addition, internal size calculations of structure members have been modified for more efficient memory use.

5.42 Reduced Memory Usage in Fetch Ahead Buffers with c-treeSQL

Fetch ahead buffer logic is a c-treeACE SQL performance feature that allows row fetches to quickly satisfy client requests. This support requires up front memory allocations to be effective.

A c-treeACE SQL configuration keyword is available to enable this feature which is now off by default. To enable this performance feature use the following configuration keyword:

```
SETENV DH_DO_AHEAD=Y
```

5.43 Reduced Memory Usage in Communications Buffers with c-treeSQL

A c-treeACE SQL performance feature involving the network communication buffer requires up front memory allocation to be effective. To avoid a constant allocation of memory by new connections, a pre-allocated static buffer had been used. In an effort to reduce memory usage, this buffer has been configured to dynamically allocate when needed, however, disabling any potential performance advantage.

5.44 Reduced Static Array Usage for Memory Efficiency in c-treeSQL

In an effort to reduce c-treeACE SQL memory usage various arrays used to project nodes and multi-row index and table fetches have been modified such that they are now dynamically allocated based upon actual size for the number of fields.

5.45 Corrected Memory Leakage in c-treeSQL ODBC Driver

A memory leak of about 1KB per connection was noted when using the c-treeSQL ODBC Driver. The driver had allocated memory for the cursor and an internal structure and had not freed these resources when no longer used. This has been corrected and the resources are properly freed when it is determined they are no longer in use.

5.46 Memory Leaked Fixed in the c-treeSQL ODBC PRESERVE CURSOR Feature

A memory leak was found in the c-treeSQL ODBC driver when using the PRESERVE CURSOR feature.

When the executing a **SQLFetch()** on a query with an empty result set, the cursor becomes closed (or not even opened), however, at the application level the odbc cursor is open. If the cursor is then closed with **SQLFreeStmt(***SQL_CLOSE***)**, as on the server side it is already closed, the logic fails with a *SQL_ERR_NOTOPENED* error and a free memory operation fails to take place.

To ensure the memory is freed, the *SQL_ERR_NOTOPENED* error is now ignored.

5.47 Avoid SKIP_MISSING_FILES with Transaction Dependent Files

A rollback using the Restore utility, **ctrdmp**, produced an unexpected **FNOP_ERR** (12). As the !SKIP attribute to the rollback script was not a good solution, it was determined that this activity should not apply to transaction dependent files.

Whenever transaction logs are used to recover, rollback (**ctrdmp**) or roll-forward (**ctfdmp**), c-treeACE scans the transaction logs to determine active transactions and to open the files that are updated. When a file cannot be opened, execution may terminate, typically with a **FNOP_ERR** (12) error. It is possible to utilize the `SKIP_MISSING_FILES` option to complete the assessment of active transactions and updated files; and the recovery/rollback/roll-forward will complete, possibly skipping operations on files that could not be opened. However, this is not always the case as some of the files that were skipped may be created or have been renamed (or deleted), and if they are transaction dependent (*TRANDEP*) files, the transaction logs contain sufficient information to permit them to be properly updated. Adding `SKIP_MISSING_FILES` means that non-*TRANDEP* files may in fact be skipped even though they should have been present.

To avoid requiring `SKIP_MISSING_FILES` when *TRANDEP* files are in use, a new default behavior effectively treats *TRANDEP* files as though `SKIP_MISSING_FILES` is turned on, however, for files without *TRANDEP* activities, recover, rollback, or roll-forward may still terminate execution if unexpected missing files are encountered.

This behavior can be turned off by adding the `COMPATIBILITY NO_AUTO_SKIP` configuration keyword to the c-treeACE configuration file, *ctsrvr.cfg*.

Note: It is possible that an unexpected **FNOP_ERR** error can still occur for a *TRANDEP* file, however, this change should greatly reduce the number of unexpected **FNOP_ERR**'s.

5.48 Detection of Missing Transaction Dependent Files During Automatic Recovery

c-treeACE is designed to permit automatic recovery and rollbacks to properly handle deleted and renamed transaction dependent files (*TRANDEP*) without requiring the `SKIP_MISSING_FILES` configuration to be enabled. When a rollback was attempted after deleting a *TRANDEP* file externally (that is, the file was deleted using a system call outside of c-tree) the rollback succeeded even though in this case there really was a missing file. The rollback improperly ignored the missing file. If `SKIP_MISSING_FILES` is active, then the rollback should succeed

even with the missing file, however, should only ignore missing files for which there is a transaction controlled explanation such as a file delete or file rename.

The following sequence of operations demonstrate this behavior:

```
TRANBEG
CREATE TRANDEP FILE
ENDTRAN
TRANBEG
ADDREC
ENDTRAN
CLOSE FILE
Delete file with a system call
Rollback to the time between Steps 3 & 4
```

The rollback encounters the missing file, however, ignores the missing file. Furthermore, this example also demonstrates that if a close and reopen is added between Steps 3 & 4, then the file is correctly detected that it is actually missing during rollback.

The recovery process has been modified such that a *CLSTRAN* log entry will attempt to open the file (if it is not already opened). Upon detecting a *CLSTRAN* entry triggers putting a missing file on the list of missing files. To revert this behavior, the server configuration keyword, `COMPATIBILITY NO_CLSTRAN_OPEN` is provided. For stand alone models, the #define NO_ctBEHAV_SKIPAUTO_CLSTRAN is available to turn off the new behavior.

5.49 Correct CTSTATUS.FCS Location with CTSTATUS_SIZE Configuration Option

The `CTSTATUS_SIZE` keyword allows control over the maximum size of the server *CTSTATUS.FCS* status log file with options to limit the number of archived files. Several observations were made when using the `CTSTATUS_SIZE` option with the `LOCAL_DIRECTORY` server configuration option.

- A new *CTSTATUS.FCS* file was created in the server executable directory;
- The *FREOPENS.FCS* stream file was not deleted on server shutdown due to a wrong path specification;
- When multiple threads were writing to the file, messages could be missing from the file;
- Logging of error messages with a non-zero error code prevented creation of a new status log file;
- A potential threading deadlock when the `DIAGNOSTICS LOWL_FILE_IO` keyword was also specified.

In the internal logging functions, the filenames and paths are now properly considered to ensure the log and stream files are created in the appropriate and expected locations. Additional modifications were done to ensure proper thread synchronizations to secure server integrity in the unlikely case of potential thread timing cases.

5.50 Passing "NULL" as a JDBC Prepared Statement Parameter

It is common to submit a NULL value for a c-treeACE SQL statement parameter in a JDBC function to bind the parameter such as the following:

```
CtreePreparedStatement.setString(x,null)
```

The c-treeACE SQL JDBC driver has been enhanced to support passing null as a value to the following supported functions:

- **CtreePreparedStatement.setBigDecimal()**
- **CtreePreparedStatement.setString()**
- **CtreePreparedStatement.setBytes()**
- **CtreePreparedStatement.setDate()**
- **CtreePreparedStatement.setTime()**
- **CtreePreparedStatement.setTimestamp()**
- **CtreePreparedStatement.setObject()**

Note: Functions not in this list are not qualified to accept the NULL value.

5.51 Proper Handling of LONG VARCHAR Fields on HIGH-LOW Architectures

An error -30008 (packet size mismatch) was reported while reading a c-treeSQL LONG VARCHAR field with the c-treeSQL ODBC driver when the data was located on an IBM AIX machine (HIGH LOW byte order architecture). A discrepancy was found in the handling of data marshalling of LVARCHAR data, as LONG field data is handled indirectly through a handle to the data. The marshalling and unmarshalling logic was modified for HIGH LOW architectures to ensure consistent handling of data on all byte-ordered architectures.

5.52 Improved Key Estimation from Partitioned Indexes

c-treeACE SQL utilizes an estimated number of keys between target values as a measure of index selectivity during query optimization. When estimating key values between two target values, an excessive bias in key distribution could be encountered with partitioned files. This could result in an inefficient index choice for a query resulting in poor performance. The **EstimateKeySpan()** and **KeyAtPercentile()** routines has been improved to provide accurate key estimations for partitioned key values.

5.53 Improved Partitioned File Creation Synchronization

When creating new file partitions, a separate temporary thread is spawned to enforce an independent transaction. It was possible in some cases that the synchronization of the new file

creation thread and the subsequent file open was not properly coordinated. In the worst case, a memory free operation could take place twice causing an abnormal server termination that was noted in extensive QA testing. The synchronization of these events is now properly managed avoiding any potential errors or exceptions.

5.54 Improved Launching of SIGNAL_READY and SIGNAL_DOWN Processes

The c-treeACE `SIGNAL_READY` and `SIGNAL_DOWN` configuration keywords allow c-treeACE to execute external processes after server startup or server shutdown respectively. When a c-treeACE process was configured with large amounts of cache memory, it was found that the `SIGNAL_READY` and `SIGNAL_DOWN` processes could silently fail. c-treeACE launches these external processes using a **fork()** system call to create a new process. As **fork()** duplicates the entire image of the server process, this can cause paging faults and in the worse case may fail if not enough swap space is available. The Unix **vfork()** system call does not duplicate the process image, and can be used when an **exec()** is immediately called in the new process, as is the case with c-treeACE.

5.55 "Transaction Persistent" Lock Support

Background

When a data record is locked and updated under transaction control, a subsequent unlock request while the transaction is still active returns as if it succeeded (i.e., NO_ERROR), however;

1. The lock is still held; and,
2. *sysiocod* is set to **UDLK_TRN** (-3).

At the time of commit or abort of the transaction, the lock will be released.

Savepoints can also cause the lock to be freed: if restoring to a savepoint removes all updates for the record, then the lock is freed whether or not an explicit unlock request was made and whether or not the *XFREED* modes are in use.

The *ctMARK_XFREED* / *ctKEEP_XFREED* transaction modes (used with **TRANBEG()** / **TRANEND()** respectively), affect the commit/abort unlock behavior by only freeing locks that were explicitly freed within the transaction.

A request to close a file that has been updated under transaction control has the close deferred until the transaction completes.

New Behavior

It is now possible to specify that locks persist even if the record has not been updated: the unlock will return **NO_ERROR** and *sysiocod* will be set to **UDLK_TRN**. A user can turn on this state for all files used by the user with a call to **ctLOKDYN(** *ctLOKDNYtranPersist* **)**. Or the user can call **PUTHDR(** *datno*, *1*, *ctTRNPERShdr* **)** for particular data files for which the new state will be activated. If *ctLOKDNYtranPersist* is not enabled, then the individual file states prevail. However, if *ctLOKDYNtranPersist* is enabled, the individual file states are ignored.

A call to **ctLOKDYN(** *ctLOKDYNnotranPersist* **)** will disable the user-wide state. A call to **PUTHDR(** *datno*, *0*, *ctTRNPERShdr* **)** will turn off the file state. The "tranPersist" state (whether user-wide or file specific) only affects behavior at the time an unlock request is made. Whether it is on or off at the time the lock is granted does not affect unlock behavior.

A call to **TRANRST()** that undoes all the updates to a record causes the lock on the record to be freed. However, the "tranPersist" state also effects this behavior. If the state is on (user-wide or file-specific) at the time of the **TRANRST()**, then the record is not unlocked when all its updates are undone.

A related issue has to do with closing a file inside of a transaction. An attempt to close a file inside a transaction that has pending record updates causes the close to be deferred until the transaction is completed. If **ctLOKDYN(** *ctLOKDYNlockDefer* **)** or **PUTHDR(** *datno*, *1*, *ctTRNPERShdr* **)** are called, then the file close is deferred if there are locks pending on the file, whether or not any records are updated. **ctLOKDYN(** *ctLOKDYNnolockDefer* **)** and **PUTHDR(** *datno*, *0*, *ctTRNPERShdr* **)** turn off this state.

The user-wide and/or file-specific routines need only be called once. The states persist until the user logs off or the file is closed, respectively. But they can be turned on and off as the application desires.

5.56 Permit Shared Reopen After a Transaction Controlled Header is Updated

Occasionally, it is quite useful to update header information in a file after it is created and before any users begin using the file. Consider the case of setting an initial serial number.

If a file is opened exclusively, and a **PUTHDR()** call updates a file attribute (such as the next serial number) under transaction control, and the file is closed and then reopened in shared mode before the transaction is committed or aborted, the following scenarios could apply:

1. Prior to this change, the reopen would fail with **FNOP_ERR** (12) / **FCNF_COD** (-8);
2. With this change the reopen succeeds, however, other users still see the file opened exclusively until the transaction is committed or aborted.

With the introduction of this open modification, other shared open attempts fail with **FNOP_ERR** (12) / **FCNF_COD** (-8) as if the file was still in exclusive mode. Once the transaction is completed (commit or abort), then other shared opens behave as expected.

5.57 Improved Handling of Header Updates During Server Shutdown

A transaction controlled file, (the c-treeACE *SYSLOGDT.FCS* file in this case) had its update flag set following auto recovery after a header write failed when closing the file. c-treeACE then failed to start with error 14 for that file.

In the case observed, c-treeACE was shutting down, and the *syslog* thread failed to write the header of *SYSLOGDT.FCS* file before closing the file. When the header write failed, the update

flag was left set on disk, however, the update flag in the file control block in memory was set to zero. As a result, the final checkpoint closed the file without attempting to write the header to disk, and the final checkpoint completed, leaving no reference to the file as an open file at shutdown.

When writing the header while closing a file, if the write fails logic is already in place to shut down c-treeACE if the file is under transaction control. New logic is introduced to set the update flag to a special value to if the update flag has been reset to zero. Doing this ensures that if a final checkpoint occurs, the final checkpoint also attempts to write the header. If the final checkpoint is able to write the header, it closes the file and the file is not marked corrupt. However, if the final checkpoint also fails to write the header, error handling code will terminate the server (**ctcatend()**), and the logic detects a recursive call to **ctcatend()** and it exits the process without writing the final checkpoint.

5.58 Two-Phase Transaction Support

Two-Phase transaction support allows, for example, a transaction to span multiple servers. This is useful for updating information from a master database to remote databases in an all-or-nothing approach.

To start a transaction that supports a two-phase commit, you would include the *ctTWOFASE* attribute in the transaction mode passed to the **Begin()** function. Call the **TRANRDY()** function to execute the first commit phase, and finally **Commit()** to execute the second commit phase.

Note: You may need additional caution with regard to locking and unlocking of records as your transactions become more complex in a multi-server environment to avoid performance problems.

Example

(Note that this example could also use individual threads of operation for the different c-tree Server connections, avoiding the c-tree instance calls.)

```
COUNT       rc1,rc2,filno1,filno2,rcc1,rcc2;
TEXTdatabuf1[128],databuf2[128];

/* Create the connections and c-tree instances */
...
if (!RegisterCtree("server_1")) {
    SwitchCtree("server_1");
    InitISAMXtd(10, 10, 64, 10, 0, "ADMIN", "ADMIN", "FAIRCOMS1");
    filno1 = OPNRFIL(0, "mydata.dat", ctSHARED);
    FirstRecord(filno1, databuf1);
    memcpy (databuf1, "new data", 8);

    /* Prepare transaction on c-tree server 1 */
    Begin(ctTRNLOG | ctTWOFASE | ctENABLE);
    ReWriteRecord(filno1, databuf2);
    rc1 = TRANRDY();
}

if (!RegisterCtree("server_2")) {
    SwitchCtree("server_2");
    InitISAMXtd(10, 10, 64, 10, 0, "ADMIN", "ADMIN", "FAIRCOMS2");
    filno2 = OPNRFIL(0, "mydata.dat", ctSHARED);
    FirstRecord(filno2, databuf2);
    memcpy (databuf2, "new data", 8);

    /* Prepare transaction on c-tree server 2 */
    Begin(ctTRNLOG | ctTWOFASE | ctENABLE);
    ReWriteRecord(filno2, databuf2);
    rc2 = TRANRDY();
}

/* Commit the transactions */

if (!rc1 && !rc2) {
    SwitchCtree("server_1");
    rcc1 = Commit(ctFREE);

    SwitchCtree("server_2");
    rcc2 = Commit(ctFREE);

    if (!rcc1 && !rcc2) {
            printf("Transaction successfully committed across both servers.\n");
    } else {
            printf("One or more units of the second commit phase of the transaction failed: rcc1=%d
rcc2=%d\n", rcc1, rcc2);
    }
} else {
    printf("One or more of the transactions failed to be prepared: rc1=%d rc2=%d\n", rc1, rc2);

    printf("Pending transactions will be aborted.\n");
    SwitchCtree("server_1");
    Abort();
    SwitchCtree("server_2");
    Abort();
```

```
}

/* Done */
SwitchCtree("server_1");
CloseISAM();
SwitchCtree("server_2");
CloseISAM();
```

Note: Two-Phase transactions can become extremely difficult to debug should there be communications problems between servers at any time during the second commit phase. This can result in out of sync data between the servers as one server may have committed while another server failed. It is always appropriate to check the return codes of the individual **Commit()** functions to ensure a complete successful transaction commit across multiple servers.

6. c-treeACE SQL Enhancements

6.1 Internal Stored Procedure Added to Verify Transaction Mode of SQL Tables

An internal stored procedure has been added to c-treeACE to verify the transaction mode of SQL tables. When *fc_check_file_tran_state* is called, it returns a list of tables (with their owner) not matching the proper transaction mode for the SQL database in use. That is, for “normal” SQL databases the list of tables not having *TRNLOG*. *fc_check_file_tran_state* requires no arguments.

6.2 Improved Shutdown Stability of the c-treeACE SQL Database Engine

Several distinct cases involving either SQL Panic conditions, or server crashes were reported during the shutdown sequence of the c-treeACE SQL database engine. A general review of the shutdown logic and analysis of the provided stack traces identified multiple critical areas that required additional logic checks to avoid generating these exceptions resulting in much improved stability in this phase of server operation.

6.3 Faster Performance for Multi-Component Indices with Parameterized Queries

When the key to search for was not known, certain logic (a function determining an "average" selectivity for an index) was written such that the function was not called if there was a multi-component index and not all components were specified. However, if this condition failed, the call was not made and a selectivity of 1 was obtained, which resulted in the index to be considered a poor choice. In fact, the index was considered a worst case scenario and avoided.

This logic has been changed such that the function is always called when the key to search for is not known, and should this call fail, then an optional default value is set instead.

6.4 Latest Support for c-treeACE SQL dbExpress

c-treeACE SQL dbExpress has been updated for RAD Studio XE.

6.5 Built-in Stored Procedure Added to Return Server Thread ID

A built in stored procedure was added to c-treeACE SQL to return the server thread ID. This procedure takes no parameters and returns a result set with a single row containing an SQL *INTEGER* column with the task ID.

```
fc_get_taskid()
```

6.6 Corrected c-treeACE SQL PANIC Resolved

c-treeACE SQL had experienced a PANIC while executing a SQL statement referencing numeric values with the following error message in *CTSTATUS.FCS*:

```
 - User# 00018       :       PANIC - DTM data_get_obj 1 PID 203
```

An internal function did not always correctly handle the case of a "xxx.0" numeric value. This was corrected by verifying that the number of decimal digits is > 0 before checking if the other digits have been used to store the sign.

6.7 Resolved Server Crash in c-treeACE SQL

A c-treeACE SQL crash was reported. A *CT_STRING* field was found to be improperly terminated within a record. Internal logic has been adjusted to ensure this condition is appropriately handled.

6.8 Server Exception with Preserved Cursor Logic Corrected

Preserver Cursor logic had introduced a situation where a NULL pointer could be de-referenced. This situation is now identified and a check is now done before such pointers are referenced.

6.9 Prevent Potential Server Exception Viewing c-treeACE SQLExecution Plans

In examining reported server crashes, additional logic was added to the memory allocation and de-allocation routines in previous server builds. A NULL pointer could be generated through this new logic. While viewing an execution plan, it was noticed one of these NULL pointers could be de-referenced attempting to free the pointer resulting in a server crash. Additional checks on NULL pointers have been placed in the memory free routines to avoid this behavior.

6.10 Prevent c-treeACE SQL Shutdown Fault

c-treeACE SQL could terminate with a SIGSEGV exception during a seemingly normal shutdown sequence.

When shutting down, c-treeACE SQL checks an internal counter to determine if any SQL threads are still active. If threads remain active, the server does not free memory allocated by c-tree when it shuts down (to prevent the active threads from crashing). However, it was found that in a recent code line a SQL connection does not properly increment the counter. To prevent the invalid count, SQL threads now correctly increment the active thread counter when logging on and decrement the active thread counter when logging off.

6.11 Avoid Communications Buffer Overrun

c-treeACE SQL was found to terminate with an unhandled exception or an internal error such as **terr** (7495).

Certain file operations such as file creates and deletes cause the server to write data to an internal ISAM communication buffer, even for SQL threads. It was possible for an SQL thread to write past the end of the buffer. At this point, an internal error or unhandled exception can occur.

The functions that write state information to the ISAM communication buffer now check if the calling thread is a SQL thread. If so, the functions return without writing to the communication buffer or changing the offset in the communication buffer.

6.12 JDBC OutOfMemory Exception Resolved

A JDBC *OutOfMemory* Exception was reported on a particular query. An internal variable had been unexpectedly not initialized in a *SQLDA* data communications packet. While the packet was correctly sized, improper interpretation of its contents could take place by the client under certain conditions. (In this particular case, a 64-bit environment compiled with optimizations.) Ensuring that all values are correctly initialized in generating the *SQLDA* packet corrects this problem.

6.13 Improved Query Performance Transforming LEFT OUTER JOINS to INNER JOINS Where Appropriate

A complex c-treeACE SQL query involving several LEFT OUTER JOINS was found to run exceptionally slow in comparison with other SQL databases. Examination of the query revealed that a transformation of some LEFT OUTER JOINS to INNER JOINS would vastly improve query efficiency. This change was enacted in two modifications:

1. If the predicate on the RHS columns of an outer join is not a IS NULL predicate, transform the outer join to an inner join and consequently move the restriction below the join node.
2. If there exists an equality join predicate in an inner join node above an outer join node, on any of the RHS columns of the outer join, then, convert the outer join into an inner join.

With these changes, a complex query that previously took longer than 50 seconds now completes in less than 1 second.

6.14 Improved Query Performance with Constant Predicate Values

A slow query was noted containing the following predicate:

```
WHERE  (user.ID IS NULL OR user.ID <> 'some_value')
```

The OR clause resulted in the query to be split and the two resulting conditions "joined" with a UNION operator. In this specific case, ID was a primary key (and thus a NOT NULL column).

Changes were made to the query optimizer to recognize a null predicate on a not-null column and mark that predicate as a constant predicate with an appropriate truth value thus improving the query performance:

- <not-null column> IS NULL always evaluates FALSE
- <not-null column> IS NOT NULL always evaluates TRUE

6.15 Avoid Duplicate Constraint Name Error

Dropping a constraint and then adding a named constraint generated a duplicate constraint name error with c-treeACE SQL. The logic used internally to generate a constraint name was based on the number of the constraint to generate the next name, however, this may cause a problem in case the constraint removed is not the last one.

New internal logic was added to return the first unused constraint ID for the table when adding a constraint to avoid this error.

6.16 Automatic Transaction Rollback Upon Deadlock Error Now Default Behavior for c-treeACE SQL

In some databases, upon a deadlock error the current transaction is expected to be automatically rolled back whereas c-treeACE SQL returns a deadlock error and maintains the current transaction state. c-treeACE SQL has been modified to conform to the generally expected behavior.

A server option is available disabling this automatic rollback behavior. Specify `SQL_OPTION NO_AUTO_ABORT` in the c-treeACE SQL configuration file *ctsrvr.cfg* to revert to the previous mode of operation.

6.17 Enhanced Prevention of Locking Deadlocks in c-treeACE SQL

An unexpected deadlock error **DEAD_ERR** (86) was returned in an application. When attempting to promote a read lock that already has a pending write lock a **DEAD_ERR** would be returned. To resolve this, the write lock is now permitted to succeed. That is, for two threads Thread A and Thread B, the read lock is successfully promoted to a write lock for Thread A, and Thread B continues to wait for its write lock.

6.18 Improved Byte Order Handling Between c-treeACE SQL Client/Server Architectures

A packet size mismatch error (-30008) occurred when reading a LONG VARCHAR column and a column size > 10,000 characters with the c-treeACE SQL ODBC driver and data was located on an IBM AIX architecture. Improper marshalling of data was found when communicating between different byte ordered systems. This logic was corrected to ensure proper handling of different byte ordered data is maintained.

6.19 Improved c-treeACE SQL Memory Efficiencies

Numerous additional areas of c-treeACE SQL memory utilizations were analyzed and several potential allocation failures have been addressed. This should lead to improved reliability of c-treeACE SQL when used in environments with low resource availability.

6.20 Correct Result Sets with Subquery Predicates in an ON Subquery

A c-treeACE SQL query had returned an unexpected rows that did not match the search criteria. The problem occurred when there was an outer join query with an OR-ed subquery predicate in the ON clause.

When there existed one or more subquery predicates in the ON clause, the query tree is generated as shown below:

```
 t1 LEFT OUTER JOIN t2 ON (subquery_predicate1 OR subquery_predicate2)
=>
(t1 LEFT OUTER JOIN t2 ON subquery_predicate1) UNION (t1 LEFT OUTER JOIN t2 ON subquery_predicate2)
```

If there exists a record in *t1* for which there is no matching record in *t2* on *subquery_predicate1*, an outer join record would be generated with NULL values for *t2*. However, later for the same record we might find a matching record in *t2* on *subquery_predicate2.* This results in projection of incorrect extra rows.

The problem has been resolved by performing the following transformation of the expression:

```
t1 LEFT OUTER JOIN t2 ON (subquery_predicate1 OR subquery_predicate2)
=>
t1 LEFT OUTER JOIN ((t1 INNER JOIN t2 ON subquery_predicate1) UNION (t1 INNER JOIN t2 ON subquery_predicate2))
```

6.21 Improved c-treeACE SQL Referential Integrity Checks

An update to a table with a foreign key into another table succeed while it should have failed. This occurred as the referenced table in the original table was locked. The lock was a blocking lock, however, an error was returned instead of waiting for the lock to clear.

The referential integrity check lock was modified such that if the fetch fails, an error is now returned from the fetch failure instead of STATUS_OK.

6.22 LONGVARCHAR AND LONGVARBINARY Support Added to c-treeACE SQL PHP

The c-treeACE SQL PHP interface did not initially support LONG VARCHAR (LVC) and LONG VARBINARY (LVB) field handling.

This support has been added and the entire LVB field is fetched from the server and returned as a binary buffer just as a typical field would be.

6.23 Corrected Pre-pended Owner Naming Convention in c-treeACE SQL

The c-treeACE SQL physical index name, by default, is pre-pended by the owner name. When `SQL_OPTION OWNER_FILE_NAMES` is not in use the physical index name on disk was found to be *owneridxname.idx* instead of *idxname.idx*

A logic error introduced when adding support for UNICODE string handling inadvertently always pre-pended the owner name to the index, even when `SQL_OPTION OWNER_FILE_NAMES` was not in use. The original and expected behavior has been restored.

6.24 ISO 8859-1 Support for c-treeACE SQL ADO.NET Data Provider

Incorrect decoding of ASCII characters > 127 was reported and it was determined that characters with an ASCII value larger than 127 were not properly converted. The standard ASCII decoder was replaced with a decoder supporting the "Latin" (iso8859-1) character set.

6.25 Better Optimization of Outer Join Transformations

A transformation of an outer join to an inner join did not take place in the case of OR-ed predicates. This OR predicate is now moved down the. expression tree for better query optimization.

6.26 Corrected Direct SQL BLOB Handling

Error 20000 was observed when calling the c-treeACE SQL Direct SQL **ctsqlSetBlob()** function when the LONG VARCHAR column was not the first parameter. An inappropriate array index was used in accessing the column. This is now corrected in the this interface.

6.27 Reduced Logging of Unnecessary c-treeACE SQL Panic Messages

While examining a *CTSTATUS.FCS* log file a Panic message was noticed during c-treeACE SQL shutdown. The Panic logic has been modified such that when a shutdown is already in progress only the error message is logged, and the thread is exited without any cleanup. The compromise behavior avoids the "Panic" message and resulting cryptic logged messages avoiding concerns to users, yet retains useful diagnostic information when needed.

6.28 Corrected Duplicate Rows Returned with JDBC Query

A simple query could return more rows than expected by duplicating some of the rows. A complex set of criteria was required for this condition to occur including operating at an isolation level greater than two (2) and a full table scan. It was found that a current ISAM position was not properly maintained during certain record fetches. Saving and restoring the proper position now avoids the duplicate row returned by the query.

6.29 Improved JDBC Updates of LONG VARCHAR Fields When no Matching Rows

A JDBC Exception, "ctree.jdbc.CtreeSQLException: CTDB - Table does not exist" was returned with c-treeACE SQL while executing an update statement that

1. updates a LONG VARCHAR (LVC) column via a parameter,
2. the where clause matches no record.

The JDBC driver LVC handling did not properly handle the case where the update statement returned no matching rows and attempted an update with invalid data. This case is now properly checked to avoid the exception.

6.30 Improved Direct ODBC Driver Linking on Solaris Operating Systems

When loading the c-treeACE SQL Direct Link ODBC Driver on Solaris, the **dlopen()** system call could fail with the message similar to the following:

```
relocation error: file /export/home/<path to file>/bin/libodbc_c.so: symbol
__1cG__CrunMex_rethrow_q6F_v_: referenced symbol not found
```

The local *RTLD_GROUP* option restricts the new libraries symbol resolution to only resolve symbols it referenced at link time, rather than symbols already loaded in the process. The direct link ODBC module was not explicitly linked with the *libCrun* option. This option is now included in the link command.

6.31 Improved Long Binary Handling with c-treeACE SQL

c-treeACE SQL Long Binary handling was found to have performance limitations due to a 8192 byte "chunking" of data across the communications network. For very large objects, this led to diminished performance. The internal **putdata()** / **getdata()** logic was modified to accept a *TPE_DT_LVB* buffer type to allow larger chunks to be utilized without compromising existing applications. This support is extended throughout the c-treeACE SQL interfaces.

As more subtle detail for Direct SQL developers, the buffer is considered "as is" and the length passed in is the significant length which is the length passed to the server. In the case of **getdata()**, the length is the maximum length of the buffer returned by the server.

6.32 Improved Unicode Handling in CodeGear 2009 c-treeACE SQL dbX Components

When executing the following statement:

```
select * from BRANCH
```

the following error was returned:

```
TDBXTypes.ZSTRING value type cannot be accessed as TDBXTypes.WideString value type
```

The *TDBXCtreeCustomMetaDataReader_2k9.TDBXCtreeIndexColumnsCursor_2k9*.**Next()** method of the c-treeACE dbX4.0 component, was attempting to set the Column Name as a *WideString* rather than an *AnsiString* type. This change has resulted with the inclusion of Unicode support in CodeGear2009. Setting the Column Name as an *AnsiString* type avoids the error.

7. c-treeACE Enhancements

7.1 Improved Parallel Performance with Distributed Cache Synchronization Controls

c-treeACE offers many advanced cache control mechanisms, including dedicated cache per file, cache priming, and many performance statistics for analysis and tuning.

While examining scalability performance on large parallel processing architectures with multitudes of CPUs it was found that c-treeACE performance saturated, indicating resource contention. In depth analysis revealed cache control contention and specifically, synchronization controls over thread access to the data and index caches.

A new approach has been taken with respect to this synchronization. Single contention points over hash lists have been replaced with arrays of mutexes, thus spreading contention across a hash list vastly improving performance in these specific cases. Further analysis revealed this same approach had wide applicability across many other single points of mutex contention with c-treeACE architecture and has been extended to these internal subsystems.

7.2 Reduced Service Control Manager Messages in CTSTATUS.FCS with c-treeACE Running as a Windows Service

When the c-tree Server was run as a Windows service, it occasionally logged numerous service status messages to the *CTSTATUS.FCS* status log file. c-treeACE was designed such that when the Service Control Manager queried the status of the service, the server logged a message to *CTSTATUS.FCS*. The server's service control handler was modified to avoid logging a message when a request is made to interrogate the service's status. Now, a message is only logged to *CTSTATUS.FCS* when and if the service is shutting down.

7.3 c-treeACE API Function to Set Distinct Key Attribute

For indices allowing duplicate keys, a count of actual distinct key values remains a useful metric. **ctDISTINCTset(***keyno,action***)** provides a means to turn on *ctDISTINCT* support for an existing index without having to rebuild the index.

ctDISTINCTset() returns the number of distinct key values in the index file *keyno*. For huge files, the return value is only the lower order word, and **ctGETHGH()** is used to retrieve the high order word. If *keyno* is opened exclusively, the value will be exact. It is permitted to call the routine with

the file opened in shared mode, but key additions and deletions made concurrently may or may not be reflected in the computed distinct count. The routine will fail if the file is opened for read only access. **ctDISTINCTset()** loops over all the key values, skipping duplicates to determine an actual distinct count.

Parameters

The *action* parameter has five options:

- *ctDNCTset* -- unconditionally compute and set the distinct count, and set the *keydup* parameter to *ctDISTINCT* (instead of *KEYDUP*). If *ctDISTINCT* was already set, this call will have the effect of replacing the estimate of the distinct count with an actual count. However subsequent additions and deletions will use the estimated approach.

  Note: the estimated approach is based on determining if the key value is distinct within the index node it occupies. This is a necessary condition for distinctness, but not sufficient when the key value occupies the first or last position in the node.

- *ctDNCTattribute* -- if *keydup* is already set to *ctDISTINCT*, return estimate; else treat the same as *ctDNCTset*. This action avoids the overhead of the actual computation of he distinct count when the file already supports the (estimated) distinct count.
- *ctDNCTcompute* -- unconditionally compute actual distinct count, but no change is made to the *keydup* parameter or the header's distinct count.
- *ctDNCTifil* -- may be OR-d into *ctDNCTset* or *ctDNCTattribute* to indicate that the *IFIL* in the associated data file should be updated automatically, if necessary, to reflect the *ctDISTINCT* support for the index.
- *ctDNCToff* -- may be combined with *ctDNCTifil* to turn off distinct count support and make the corresponding update to the associated *IFIL* structure. As with *ctDNCTset*, either an index file number or a data file number may be passed. If an index file number is used, then only that index is updated. If a data file number is used, then all associated indices that support a distinct count are updated.

A data file number may be passed in which case all associated indices that support duplicate key values have the requested action performed on them, and instead of returning the computed number of distinct key values, the number of indices supporting duplicates is returned.

Return Codes

If an error occurs, **ctDISTINCTset()** returns zero. On a zero return, *uerr_cod* must be checked to see if an error occurred or the index is empty.

Transaction Control

If **ctDISTINCTset()** is called for a transaction controlled file, then either a transaction is started if none is active, or an internal savepoint is started. Upon successful completion of the routine, the transaction is committed or the savepoint is cleared. In the event an error occurs, the transaction is aborted or the savepoint is restored. The new *ctDISTINCT* attribute is not set until the transaction is committed, either by the transaction that was already active when the call was made or by the one started by **ctDISTINCTset()** itself. This means that if updates are made before the commit, the distinct count will not reflect them. This is permitted since ultimately the distinct count is considered an estimate, not an exact value.

If more than one thread attempts to call **ctDISTINCTset()** for the same index, only the first is permitted. The subsequent calls receive error **DNCT_XCL** (846).

7.4 Configurable Disk I/O Sizes on Windows OS

A large variable-length record write operation could fail with Windows error 1450 ("Insufficient system resources exist to complete the requested service") when the data file resided on a network drive. c-tree normally writes the record in a single call to the **WriteFile()** Win32 API function. To avoid this error, c-tree supports setting a maximum disk read and write size. Read or write operations that exceed this size are performed as a series of reads or writes of the specified maximum size.

A configurable option is available to explicitly set the disk I/O size to use: *IO_BLOCK_SIZE*.

c-treeACE Server

Use the c-tree Server configuration keyword `IO_BLOCK_SIZE` to split disk read and write operations larger than the specified size into individual operations of the specified size. For example, specifying `IO_BLOCK_SIZE 16 KB` causes a 1 MB write request to be performed as 64 16 KB write operations.

Standalone Models

To set a maximum disk read and write size, compile c-tree Plus with #define ctMAX_IO_SIZE <maxbytes>, where *<maxbytes>* is the maximum number of bytes c-tree will read or write in a single system call.

Note: #define ctMAX_IO_SIZE is supported on Windows systems only and has no effect on other platforms.

I/O Block Sizes with Windows Systems

When the `IO_ERROR_BLOCK_SIZE` configuration option is specified in the FairCom Server configuration file, if a disk read or write operation fails with system error 1450 (Insufficient system resources exist to complete the requested service), the server retries the I/O operation using the specified block size. If the retried I/O operation also fails with error 1450 (or if a disk I/O operation fails with error 1450 and `IO_ERROR_BLOCK_SIZE` is not specified in the server configuration file), c-treeACE now logs the following message to *CTSTATUS.FCS*:

```
<op_code>: loc <location> file <filename> offset <offset> iosize <iosize> syserr <errcod>
```

where:

- *<op_code>* is **READ_ERR** (36, indicating that a read operation failed) or **WRITE_ERR** (37, indicating that a write operation failed)
- *<location>* is the location in the code where the I/O operation failed
- *<filename>* is the name of the file for which the I/O operation was requested
- *<offset>* is the offset of the failed I/O operation
- *<iosize>* is the size of the failed I/O operation
- *<errcod>* is the system error code for the failed I/O operation

An internal write call was modified to ensure that when the `IO_ERROR_BLOCK_SIZE` configuration option is used, a retried I/O operation is done at the original offset for the I/O operation and that *sysiocod* is reset to zero before retrying the I/O operation.

Avoid Invalid Error Returns with Maximum I/O Size Disk Operations

When the c-tree standalone library is compiled on Windows with #define ctMAX_IO_SIZE set to a maximum disk read/write chunk size, a c-tree API call that reads data from disk may unexpectedly fail with error **REXT_ERR** (748, read failed external cause).

A read operation near the end of a file can succeed even though it reads fewer bytes than requested. c-tree detects this case by setting the c-tree system error code variable, *sysiocod*, to zero before performing the read operation. The expectation is that c-tree's read function sets *sysiocod* to a non-zero value only if the read operation fails. However, the ctMAX_IO_SIZE logic sets *sysiocod* to the last Windows error code value even when the read system call succeeds but fewer bytes were read than were requested. The problem is that after a successful read call, the last Windows error code value could still be set to a non-zero value from a previous Win32 API call error, which causes the *sysiocod* to be set to a non-zero value even for a successful read near the end of the file.

The ctMAX_IO_SIZE logic was modified to only set *sysiocod* if the read (or write) system call fails. This is consistent with normal c-tree logic.

7.5 Mask Routine Backup Messages in CTSTATUS.FCS

Normally, a dynamic dump writes the names of all the files it backs up to the FairCom Server status log, *CTSTATUS.FCS*. The c-treeACE configuration option:

```
CTSTATUS_MASK DYNAMIC_DUMP_FILES
```

can be used to suppress the logging of the names of the files backed up by a dynamic dump operation. This option reduces the amount of information logged in *CTSTATUS.FCS* for easier analysis by an administrator.

Run Time Configuration

The **ctSETCFG()** function can be used to dynamically turn this option on or off while c-treeACE is running.

Examples

To turn the option on:

```
    ctSETCFG("setcfgCTSTATUS_MASK", "DYNAMIC_DUMP_FILES");
```

To turn the option off:

```
    ctSETCFG("setcfgCTSTATUS_MASK", "~DYNAMIC_DUMP_FILES");
```

7.6 Elimination of Strict Serialization Warning

An unexpected warning message was output to the system console.

```
ctSS redlzr warning
```

This warning message should not appear as strict serialization was not in use. Further, the warning would be unexpected even if strict serialization had been in force. The strict serializer logic was examined to be sure that it is invoked only under the proper conditions (i.e., a read lock and write lock will coexist and this is the first instance of such coexistence for the given file). Further, the logic was also examined and changed to be sure only the read serializer could acquire more coexistent locks.

7.7 Correct ctunf1 File Reformat Utility Byte Alignment on HP-UX

A byte alignment issue was found in the HP-UX build of the **ctunf1** File Reformat utility that resulted in sporadic crashing of the utility. Proper compiler settings and structure alignments are now in place to prevent this behavior.

7.8 Corrected Syntax Error When Compiling Standalone Models

When compiling standalone code without `#define ctPortCONSOLE_APP` a syntax error occurred.

```
Compile error: ctsw32_w.c(224) : error C2143: syntax error : missing ')' before '%'
```

The missing quote was added and a minor code rearrangement was done to properly fix this error.

7.9 Change Response to FUNK_ERR(13) During Automatic Recovery

A **FUNK_ERR** (13, file unknown or corrupted) was noted at the beginning of automatic recovery. The first phase of automatic recovery is to forward scan transaction logs starting from the last checkpoint. When the scan encounters an *ENDLOG* entry in a log, it closes the current log and opens the next log. If **FUNK_ERR** occurs while trying to open and read the header of the next log, recovery is terminated. As **FUNK_ERR** error indicates a bad header, new logic was added that renames the file with the bad header, and constructs a new log file to replace it permitting recovery to continue and the end of recovery checkpoint will be written into the new log.

Note: The downside to this approach is if the log with the bad header includes valid log entries. If such valid entries exist, then automatic recovery will not include any of these or subsequent log entries.

7.10 Corrected LEOF_ERR After Automatic Recovery

Error **LEOF_ERR** (30) was noted after automatic recovery during testing, where presumably automatic recovery had been interrupted just after the recovery checkpoint had been written to the log, but just before the *S*.FCS* file was updated to point to this new checkpoint.

At the restart of the application, two important points should be noted:

1. *S*.FCS* did not point to the latest checkpoint
2. All index and data files had been successfully recovered

When the application was restarted, the *S*.FCS* files indicated the next to last checkpoint as the start point for recovery. This caused the already-recovered files to be opened, and their headers possibly updated to reflect the contents of the "old" checkpoint. This should be benign, however, the recovery then encounters the actual last checkpoint causing recovery to close all files, reset all internal states, and begin recovery from this more current checkpoint. This is intended to happen except that the closing of all the files results in the header that was updated by the old checkpoint to be written to disk. At this point the header is now out of date, and not consistent with the "real" last checkpoint.

New logic was added such that when an unexpected checkpoint is encountered during recovery, all files are closed, however, the files' headers are not permitted to be written to disk during the close.

7.11 Improved Automatic Recovery in Single-User TRANRPROC Mode

Automatic recovery in a single-user *TRANPROC* application was found to not correctly reconstruct a record image (0xffs were found where data should have been). The checkpoint logic that ensures a file's disk image is flushed to disk did not properly handle non-server compiles and certain save calls were skipped. These calls are now done for updated files during a checkpoint in the critical logic that handles flushing.

Note: For single-user *TRANPROC* applications on Windows operating systems, this change should be considered a critical fix. Contact your nearest FairCom office for a simple change to a code module to enact this change in your application builds.

7.12 Resolved KLNK_ERR During Index Rebuild

A **KLNK_ERR** (25) error was noted in *CTSTATUS.FCS* during a rebuild operation.

```
Thu Oct 16 12:25:22 2008
 - User# 00001 KLNK_ERR: pos=1400x predsr=0x
Thu Oct 16 12:25:22 2008
 - User# 00001 Detected error in deleted node list, fixed list and retrying new node operation: 25
```

An internal function had computed the wrong starting offset for scanning the index file when the file was not a huge file, however, had extended header support enabled. This call was changed to

properly compute the offset for starting the index scan and now accounts for the additional size of the extended header.

7.13 Corrected Build of c-treeACE .NET Client Assembly

A syntax error was reported while building a client side .NET interface. The *FairCom.CtreeDb.dll* assembly was included by the linker, however, was not needed. mtmake was modified to properly build this client .NET model.

7.14 Option to Disable Auto TRNLOG Messages in CTSTATUS.FCS

The c-treeACE configuration option `CTSTATUS_MASK WARNING_AUTO_TRNLOG` suppresses the logging of the message:

```
"WARNING: could not turn on AUTO_PREIMG/AUTO_TRNLOG for ... (filename)"
```

when the `AUTO_PREIMG` or `AUTO_TRNLOG` configuration option matches the name of a c-tree index file that was not created with support for transaction control.

Dynamic Configuration Options

All the `CTSTATUS_MASK` options can now be enabled and disabled using the **ctSETCFG()** API function. To indicate that an option is to be disabled, prefix the option value with a tilde (~).

Example

```
/* Suppress ATUO_TRNLOG warning messages. */
ctSETCFG(setcfgCTSTATUS_MASK, "WARNING_AUTO_TRNLOG");

/* Do not suppress ATUO_TRNLOG warning messages. */
ctSETCFG(setcfgCTSTATUS_MASK, "~WARNING_AUTO_TRNLOG");
```

8. c-treeDB Enhancements

8.1 Distinct Keys Index Feature in FairCom DB API

To optimize some SQL queries a better estimation of the duplicate index selectivity is required, which implies determining how many distinct keys there are in an index allowing duplicates. Support for this feature is available in the core c-treeACE database engine. Support has been added for FairCom DB API to take advantage of this feature.

Two new functions were added to FairCom DB API to set and retrieve the distinct count flag status for an index:

```
CTDBRET ctdbSetIndexDistinctKeyCountFlag(CTHANDLE Handle, CTBOOL lag);
CTBOOL  ctdbGetIndexDistinctKeyCountFlag(CTHANDLE Handle);
```

With FairCom DB API it is also possible to "promote" a duplicate index to a duplicate index with a distinct count by opening the table, calling **ctdbSetIndexDistinctKeyCountFlag()** for the index and finally calling **ctdbAlterTable()** to execute the changes.

In promoting a duplicate index to include the c-tree distinct key count feature a new mode was added to FairCom DB API. The *CTDBINDEX_DUPCNTFLAG* index status value indicates that the index has distinct key count capability. The FairCom DB API **ctdbSetIndexDistinctKeyCountFlag()** function was modified to set this new mode. This avoids having to rebuild the index as it instead counts the distinct keys and updates the *IFIL* and index file mode information.

ctdbAlterTable() logic was also modified to detect if the only change is *CTDBINDEX_DUPCNTFLAG* and act accordingly on the affected indexes.

A FairCom DB API utility, **ctdbdistinct**, is available to enable the distinct key count feature for existing index files and this utility is created from **mtmake** by default.

8.2 Unsigned 64 bit Integer Support Added

FairCom DB API C and C++ now include support for unsigned 64 bit integers as *CT_INT8U*.

8.3 Correct Key Transformations with c-treeDB

Using the **ctdbFindTarget()** function to find a key when the key was already transformed result in error **INOT_ERR** (101). **ctdbFindTarget()** could internally call the c-tree key transformation function **TRFMKEY()** on the key passed in, thus inadvertently transforming an already transformed key value.

ctdbFindTarget() as documented does not require a transformed key. The internal call to transform the key has been removed and it is expected now that any key passed in is already transformed (if necessary) with the **ctdbBuildTargetKey()** function.

Note: This change could introduce a backward compatibility issue for some customers using **ctdbFindTarget()** or **ctdbFindTargetKey()**. `#define CTDB_FIND_TFRMKEY` can be turned on to revert to the original behavior.

8.4 ctdbWriteRecord Now Ignores Unlocking New Records

The c-treeDB **ctdbWriteRecord()** function failed with a **UDLK_ERR** error (41, could not unlock record) when adding a new record and the automatic transaction mode (*AUTO_TRNLOG*) was enabled for the file.

When adding a record to an *AUTO_TRNLOG* (or *AUTO_PREIMG*) file, c-tree automatically begins a transaction if no transaction is active. c-tree then adds the record and commits the transaction, freeing the lock on the newly-added record. However, **ctdbWriteRecord()** calls **LOKREC()** to free the record lock after adding the record, and this call failed with a **UDLK_ERR** error as the record is no longer locked. **ctdbWriteRecord()** now ignores the **UDLK_ERR** when attempting to unlock a newly-added record.

8.5 Prevent FairCom DB API Alter Table ISLN _ERR (115) Errors with Index Segment Changes

The FairCom DB API **ctdbAlterTable()** call on an ISAM created table which contained index segments 1 byte shorter than the field length unexpectedly failed with an **ISLN_ERR** error (115, key segments do not match key length). A FairCom DB API internal segment count was improperly incremented, even though the new segments were not added. The internal FairCom DB APImaintenance of segment changes was modified to resolve this error.

8.6 Retain non-Mandatory Batch Modes with the FairCom DB API Set Batch Function

The **ctdbSetBatch()** function did not properly delete records when called with a mode of *CTBATCH_DEL* | *CTBATCH_LKEY* as it seemed the non-mandatory mode of *CTBATCH_LKEY* was ignored.

The non-mandatory c-treeDB mode values do not directly map to c-tree values and are instead derived. In **ctdbSetBatch()** a *ULONG* value was cast to a *COUNT* value losing information in the conversion. A proper cast of values prevents this loss of batch mode information.

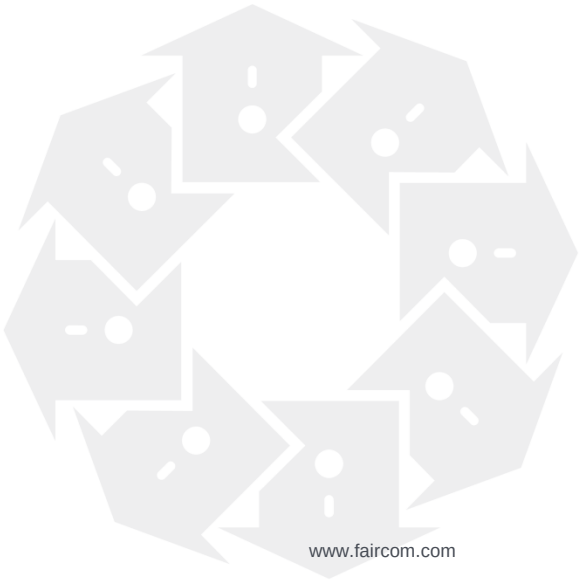
9. c-treeACE VCL Enhancements

9.1 New VCL Node Name Support

The Node Name allows a c-treeACE client to provide additional identifying information to the server. The c-treeACE VCL components now support client side Node Name support.

9.2 New CodeGear 2009 Support

c-treeACE VCL Components have been updated for CodeGear 2009.



Copyright Notice

Trademarks

FairCom DB, FairCom EDGE, c-treeRTG, c-treeACE, c-treeAMS, c-treeEDGE, c-tree Plus, c-tree, r-tree, FairCom, and FairCom's circular disc logo are trademarks of FairCom USA, registered in the United States and other countries.

The following are third-party trademarks: Btrieve is a registered trademark of Actian Corporation. Amazon Web Services, the "Powered by AWS" logo, and AWS are trademarks of Amazon.com, Inc. or its affiliates in the United States and/or other countries. AMD and AMD Opteron are trademarks of Advanced Micro Devices, Inc. Macintosh, Mac, Mac OS, and Xcode are trademarks of Apple Inc., registered in the U.S. and other countries. Embarcadero, the Embarcadero Technologies logos and all other Embarcadero Technologies product or service names are trademarks, service marks, and/or registered trademarks of Embarcadero Technologies, Inc. and are protected by the laws of the United States and other countries. HP and HP-UX are registered trademarks of the Hewlett-Packard Company. AIX, IBM, POWER6, POWER7, POWER8, POWER9, POWER10 and pSeries are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. Intel, Intel Core, Itanium, Pentium and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. ACUCOBOL-GT, Micro Focus, RM/COBOL, and Visual COBOL are trademarks or registered trademarks of Micro Focus (IP) Limited or its subsidiaries in the United Kingdom, United States and other countries. Microsoft, the .NET logo, the Windows logo, Access, Excel, SQL Server, Visual Basic, Visual C++, Visual C#, Visual Studio, Windows, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Oracle and Java are registered trademarks of Oracle and/or its affiliates. QNX and Neutrino are registered trademarks of QNX Software Systems Ltd. in certain jurisdictions. CentOS, Red Hat, and the Shadow Man logo are registered trademarks of Red Hat, Inc. in the United States and other countries, used with permission. SAP® Business Objects, SAP® Crystal Reports and SAP® BusinessObjects™ Web Intelligence® as well as their respective logos are trademarks or registered trademarks of SAP. SUSE" and the SUSE logo are trademarks of SUSE LLC or its subsidiaries or affiliates. UNIX and UNIXWARE are registered trademarks of The Open Group in the United States and other countries. Linux is a trademark of Linus Torvalds in the United States, other countries, or both. Python and PyCon are trademarks or registered trademarks of the Python Software Foundation. isCOBOL and Veryant are trademarks or registered trademarks of Veryant in the United States and other countries. OpenServer is a trademark or registered trademark of Xinuos, Inc. in the U.S.A. and other countries. Unicode and the Unicode Logo are registered trademarks of Unicode, Inc. in the United States and other countries.

All other trademarks, trade names, company names, product names, and registered trademarks are the property of their respective holders.

Open Source Components

Like most software development companies, FairCom uses third-party components to provide some functionality within our technology. Often those third-party components are selected because they are a standard in the industry, they offer specific functionality that is easier to license than to develop and maintain in the long run, or they provide a proven and inexpensive solution to a particular business need. Examples of third-party software FairCom uses are the OpenSSL toolkit that provides Transport Layer Security (TLS) for secure communications and the ICU Unicode libraries to provide wide character support (think international characters and emojis).

Some of these third-party components are the subject to commercial licenses and others are subject to open source licenses. For open source solutions that we incorporate into our technology, we include the package name and associated license in a notice.txt file found in the same directory as the server.

The notice.txt file should always stay in the same directory as the server. This is particularly important in instances where your company has redistribution rights, such as an ISV who duplicates server binaries and (re)distributes those to an eventual end-user at a third-party company. Ensuring that the notice.txt file "travels with" the server binary is important to maintain third-party and FairCom license compliance.

2/5/2025

10. Index