test link

Release Notes

FairCom DB V12 Release Notes

Audience

Developers

Subject

V12 Updates and Changes to the FairCom Database

FairCom®

Contents

1. FairCom DB V12 Release Notes

This document lists many fixes that have been made to improve the stability of FairCom DB.

Be sure to also see the FairCom DB V12 Update Guide (https://docs.faircom.com/doc/v12-update-guide/) for important details, especially the Compatibility Notes (https://docs.faircom.com/doc/v12-update-guide/CompatibilityNotes.htm) chapter.

2. Critical Fixes

Note this section lists the more severe fixes. Additional fixes that are deemed to be less severe are documented throughout the remainder of this guide.

2.1 Automatic recovery and duplicate file IDs

If c-tree data or index files that are under full transaction control are copied and then accessed by the same FairCom Server that is accessing the original files, automatic recovery might apply changes to the wrong copy of the file.

FairCom Server stores a file ID that is used during normal operation and during automatic recovery. It is able to reassign the file ID of the copied file if the original file is open when the copied file is accessed. If the two copies of the file were not open at the same time, the duplicate file ID was not detected, which could cause automatic recovery to apply changes to the wrong file.

FairCom RTG includes a **ctutil** *-fileid* command to update the file ID of FairCom RTG files.

Note: Copying files is against FairCom's recommended best practices. However, because copying files is fairly common in many industries, FairCom has the following provisions:

1. **ctutil -filecopy and -copy options** - Use the **ctutil** *-filecopy* and *-copy* options to copy files.
2. **Restamp file ID after copying using the operating system** - The **ctfileid** Update File ID utility (*-fileid*) is for restamping the file ID after a file is copied using the operating system facilities. It is located in *tools\cmdline*. See the *Command-Line Tools* documentation: *ctfileid - Update File IDs* (*https://docs.faircom.com/doc/cmdline/ctfileid-util.htm*).

Note: To avoid any further recovery issues with matching file IDs, FairCom strongly suggests educating your customer base to use option 1 to copy files or, after copying a file using option 2.

Automatic recovery handling of duplicate file IDs and transaction-dependent operations

In cases involving transaction-dependent file creates and renames, a duplicate file ID is caused by a system copy of a file, which could cause problems during automatic recovery. Symptoms included recovery failing with internal error **8777**, or record counts being incorrect after automatic recovery, or possibly even an unhandled exception in very obscure situations. If automatic recovery opened the renamed file and the original file, their file IDs could conflict, causing internal error **8777**. Or the redo of the create could cause the record count in the data file header to be reset to zero, or adjusted for operations on the copied or renamed file.

Automatic recovery has been modified to address this situation.

Improved detection of logically equivalent filenames during automatic recovery

Opening a file using a different, but logically equivalent path specification could prevent automatic recovery from determining that the file was part of a transaction-dependent rename operation. This could happen with two different physical files that have the same file ID. The filename check for transaction-dependent file operations has been improved so the logically equivalent path specifications are determined to be for the same file.

2.2 Improved recovery handling of system file copy followed by deleting the file

Automatic recovery may fail with error **8777** when a recently-created file is copied using a system command and then deleted using c-tree. Automatic recovery may try to redo the create of the file. The logic has been changed to address this situation.

2.3 Automatic recovery failure with code L64 or error 1120 with unusual index conditions

Automatic recovery may fail with an **L64** internal error or error **1120** if an index is re-created with more members than the original index had and automatic recovery needed to act on log entries before the new member was added. The logic has been modified to properly handle this situation.

2.4 Rare automatic recovery failure fixed

In a rare situation, FairCom DB automatic recovery could fail with error **26** when a LOGEXTFIL entry's transaction file number was unexpectedly zero. The error would occur after the following message is logged to *CTSTATUS.FCS*:

```
WARNING: zero log-entry file handle (tfil)
```

The logic has been modified to correct this.

2.5 Turning background flush for TRNLOG files off and on may cause memory overwrite

Although not something typically seen in production, turning background flush for TRNLOG files off and on could cause a memory overwrite. For example, using **ctadmn** to disable and then enable the `TRAN_DATA_FLUSH_SEC` keyword can cause this problem. The logic has been modified to correct this problem.

2.6 c-tree Server no longer terminates with unhandled exception when DISK_FULL_ACTION causes shutdown

When the `DISK_FULL_ACTION` server configuration option was used and the available disk space became lower than the specified threshold, causing the server to shut down, the server process terminated with an unhandled exception. The logic has been modified to correct this.

2.7 Rare missing index keys after Create or Rebuild fixed

In certain specific circumstances (see below), one or more index members had fewer keys than expected after an index rebuild or creation. The creation/rebuild returned successfully, which sometimes led to other unexpected behavior.

The circumstances for this problem to be encountered required both of the following: a configuration with `SORT_MEMORY 6GB` or larger and the logical index being rebuilt large enough to be swapped to disk.

The logic has been modified to eliminate this.

In addition, logic was implemented to validate the number of keys added for sorting is the same as the number of keys returned. A discrepancy returns new error **ISORT_ERR** (1135).

This update will not fix an existing index with missing keys. The index will need to be rebuilt after the update or as a workaround it can be rebuilt in older versions with `SORT_MEMORY` set to less than `6GB`.

2.8 Enhanced Detection and Reporting of Filesystem Flush Errors to Protect Data Integrity

Potential silent data loss following fsync() failures

When using buffered filesystem I/O for data and index files (FairCom DB default), an I/O error during a filesystem flush request (e.g. **fsync()**) could result in data loss depending on the filesystem implementation details, even if a future flush request succeeded (as filesystem contents may be invalidated and released upon failure before the next flush attempt). In addition, while committed transactions were secured to the transaction log, transaction log data may no longer have been available for recovery upon an assumed successful data/index flush state resulting in data/index files in a permanently unrecoverable data loss state.

For files under full transaction control, such an error can be treated identically to a failed write error, which is considered fatal in most situations. It is considered safe to terminate on any failed I/O write or flush of transaction controlled data/index files, because all committed transactions have been secured to the logs.

Behavior Change: Starting with code lines dated with a Build after 190314, an **fsync()** error will now behave similar to a failed write, which is expected to be a fatal error for a ctTRNLOG file and will cause the FairCom Server to shut down.

Workaround: Using **all** the following *ctsrvr.cfg* configuration options should avoid this issue for files under transaction control by opening them in a synchronous write mode:

`COMPATIBILITY LOG_WRITETHRU` is the recommended transaction log write mode. It is recommended to not change or disable this mode.

`COMPATIBILITY TDATA_WRITETHRU` is strongly recommended for all transaction controlled files and is considered to perform well.

`COMPATIBILITY TINDEX_WRITETHRU` is the safest option, however it may impact performance and performance testing should be done to ensure acceptable throughput is maintained.

Systems that have little idle disk time or that have large c-tree caches relative to system RAM will benefit the most from using these keywords.

Note: The existing `COMPATIBILITY CHKPNT_FLUSHALL` configuration option should no longer be considered for use in production environments due to long-term potential data integrity issues as the **sync()** system call does not detect filesystem write errors.

2.9 SQL - Rare server crash after network error fixed

A server crash occurred while freeing memory after a network error was reported. The logic has been modified to correct this unexpected situation.

2.10 Improved Detection of Logically Equivalent Filenames during Automatic Recovery

Opening a file using a different, but logically equivalent path specification could prevent automatic recovery from determining that the file was part of a transaction-dependent rename operation. This could happen with two different physical files that have the same file ID. The filename check for transaction-dependent file operations has been improved so the logically equivalent path specifications are determined to be for the same file.

2.11 Fixed rare server crash following errors during index node split

An unusual error such as a system I/O error during an index node split could cause the server to terminate. This affects server versions V10.3 and later. *CTSTATUS* would contain a termination message similar to the following:

```
- User# 00118 O118 M18 L81 F1 P0x (recur #1) (uerr_cod=36)
```

The logic has been modified to correct this problem.

2.12 Fixed crash and recovery errors due to transaction logs exceeding 2GB

Inserting a 1.8GB variable-length record into a transaction enabled file caused a c-tree Server crash with an "Unexpected internal c-tree(R) error #**8520**". Then during automatic recovery an error **96** was reported and the automatic recovery was aborted. Inserting the 1.8GB record caused the transaction log size to exceed the 2GB maximum we allow. The logic has been

carefully reviewed and corrected to resolve this situation. c-tree's maximum variable-length record size of 2GB is fully supported in transaction enabled files.

Workaround: This issue could affect a server with `RECOVER_MEMLOG` enabled or utilities such as **ctldmp**, **ctrdmp**, and **ctfdmp** that internally enable this functionality. Each memlog allocation is limited to under 2GB. Disabling the `RECOVER_MEMLOG` functionality could provide a workaround for this issue.

Note: The maximum allowed value for the `LOG_SPACE` keyword has been increased from 1GB to 4GB-1. This controls the initial size of transaction logs (of 4 logs specifically), so each log can now grow to just under 1GB. Each individual log can now grow up to 4GB if needed, although 3GB is expected to be the largest size, which will be a 1GB normal log size, plus a 2GB variable-length record.

2.13 Fixed server crash from corrupted Resource Definition Map

A server crash could occur when trying to open a data file with a corrupted DODA resource definition map. The following was logged in *CTSTATUS*:

```
User# 00021 Exception Handler: An unhandled exception 0xc0000005 occurred at address 00007FFB2FC38270.
```

The logic has been modified to correct this.

A new error code, **RCPT_ERR** - 1156, has been added. It is returned when this type of corruption in the resource definition map is detected.

2.14 Server no longer crashes if ctSysQueueClose() is called while queue in use

The FairCom Server could crash if **ctSysQueueClose()** was called while another user is still accessing the queue.

A **ctSysQueue** handle can be closed and the same handle reassigned to a new queue. Users of the original handle may not be aware that the handle now references a different queue, potentially resulting in unexpected messages in the new queue. This behavior has now been corrected.

Compatibility Change: This is a compatibility change in the client/server communication API. Old client libraries accessing a server with this change will receive error **QCVR_ERR** (1123) if making a **ctSysQueue** API call. A client with this change accessing an older server will receive error **QSVR_ERR** (1124) if making a **ctSysQueue** API call.

2.15 Memory corruption during multi-member index file open fixed

A server crash or other unexpected behavior could be observed during file open on an index that has more than one member. The logic has been modified to correct this.

2.16 SQL crash after PANIC with unusual query

A server crash was seen while running a SQL query. *CTSTATUS.FCS* had the following message:

```
PANIC - XEC et_pexpr::xec_set_datap 1
```

This error message has been expanded to provide additional information in the event it is encountered again.

Note: This modification propagates errors back to the application in this case avoiding a crash. Observed errors should be captured by the application for analysis.

2.17 SQL columns defined as money(17) or money(18) no longer silently overflow

There are now two types of Money with the following syntax: `money (prec, scale)`.

The following are the possible values and resulting rounding behavior.

- `money(32,2)` - round to 2 digits
- `money(32,4)` - round to 4 digits

Attempting insert into a column defined as `money(17)` or `money(18)` could result in:

1. no error but the value stored and then retrieved was not correct
2. **error(-21038): CTDB - Operation causes Overflow**

The logic has been modified to correct these problems.

For customers using `money(17)` and `money(18)`, problem #1 does not occur anymore. With this modification, the insert/update fails with **error(-21038)**, because the mapping cannot be changed for an existing table (having existing data).

Affected Components: SQL Server

Compatibility: This modification breaks compatibility on the field type mapping for `money(17)` and `money(18)`. This is an issue for people creating a table in SQL and accessing it in ISAM if either of the two mentioned precisions is used.

2.18 Unusual SQL query no longer causes panic and server crash

A SQL query was causing a panic message "`PANIC - TPEUTIL vl:operator []`" to be logged in *CTSTATUS.FCS* followed by a server crash. The logic has been modified to avoid the panic and the server crash.

3. Notable Compatibility Changes

Note this section lists the more severe compatibility changes. To see all compatibility changes, search this document for "compatibility change". Also search for "behavior change". A behavior change is viewed to be less severe than a "compatibility change".

3.1 FairCom Server file permission inheritance changes

We have cleaned up server file permission inheritance. Before this, setting permissions on files and then retrieving them could show permissions that were not explicitly set in the file permission mask. Taking into account c-tree's file permission inheritance hierarchy, the results could be unexpected in some cases. Here are some examples using the **sa_admin** utility to set file permissions:

Example #1:

Set permission:

```
OPF_WRITE GPF_DELETE WPF_READ
sa_admin -aADMIN -pADMIN -f"" -sFAIRCOMS -ofs mark.dat "-+------+-+----"
```

Resulting permission:

```
OPF_READ OPF_WRITE OPF_DELETE GPF_READ GPF_DELETE WPF_READ
```

Expected effective permission:

```
OPF_READ OPF_WRITE GPF_READ GPF_DELETE WPF_READ
```

(owner should not inherit DELETE from group)

Example #2:

Set permission:

```
OPF_WRITE WPF_READ
sa_admin -aADMIN -pADMIN -f"" -sFAIRCOMS -ofs mark.dat "-+--------+----"
```

Resulting permission:

```
OPF_WRITE WPF_READ
```

Expected effective permission:

```
OPF_READ OPF_WRITE GPF_READ WPF_READ
```

(owner and group should inherit READ from world even if owner or group have no other permissions)

Example #3:

Set permission:

```
OPF_READ GPF_DELETE WPF_WRITE
sa_admin -aADMIN -pADMIN -f"" -sFAIRCOMS -ofs mark.dat "+-------+--+---"
```

Resulting permission:

```
OPF_READ OPF_DELETE GPF_DELETE WPF_WRITE
```

Expected effective permission:

```
OPF_READ OPF_WRITE GPF_WRITE GPF_DELETE WPF_WRITE
```

(owner should not inherit DELETE from group; owner and group should inherit WRITE from world)

Causes

The following reasons explain this behavior:

1. The group inherits the world permissions and the owner inherits the group permissions.
2. When setting permissions on a file, the inherited permissions are applied to the permission mask that is stored in the file rather than being applied when reading the permissions from the file based on the permission inheritance hierarchy.
3. An improper bitmask macro definition caused the world WRITE permission to not be inherited by group or owner.
4. If group has the GPF_NONE permission specified, no inheritance of permissions from world to group occurs. If owner has the OPF_NONE permission specified, no inheritance of permissions from group to owner occurs. The **sa_admin** and **ctadmn** utilities turn on the GPF_NONE or OPF_NONE bit if no other group or owner permission is specified, which explains why in example #2 the world READ permission is not inherited by group or owner.

Changes

To improve the file permission inheritance behavior, the following changes have been made:

1. Permission inheritance is now calculated when reading the permissions from the file, so the permissions as originally specified are written to the file.
2. Inheritance occurs from world to group and from world to owner. Inheritance from group to owner no longer occurs, as this has been determined to be inappropriate.
3. Inheritance to group and owner occurs even if the GPF_NONE or OPF_NONE permission is specified.
4. The bitmask was corrected so that world WRITE permission is properly inherited by group and owner.

Note: If a file had a permission improperly inherited from group to owner, as in examples #1 and #3 above, this modification does not change that permission, because that permission value is stored in the file. However, for a file having permissions that were not inherited but should have been, as in example #2, the code modifications correct this situation by evaluating the permission inheritance when opening the file.

Compatibility Notes:

Behavior Change: This change modifies c-tree Server's inheritance of file permissions. As a result, behavior of file permissions differs from its previous behavior in the following ways, where "old server" means c-tree Server with original file permission behavior and "new server" means c-tree Server with the modified file permission behavior:

- The old server modified the file permissions based on the permission inheritance rules before storing the permissions in the file, so retrieving the permissions showed the inherited permissions. The new server stores the file permissions as originally specified by the administrator and calculates the permission inheritance when opening the file.
- When a new server opens a file whose permission was set by an old server, the only potential change in effective permissions is that additional permissions might be inherited from world to group and world to owner that were not inherited by the old server.
- When the new server sets permissions on a file, reading the permissions will return the exact same permissions that were set, and inheritance of permissions from group to owner does not occur.

Here are some specific cases to demonstrate this behavior:

Case #1:

Old Behavior: For a file that has world permissions assigned and no group permissions assigned, group and owner did not inherit any permissions.

New Behavior: For a file that has world permissions assigned (regardless of group permission assignment), group inherits world permissions, and owner inherits world permissions.

In this case, group and owner are granted additional permissions (the world permissions) compared to the old behavior.

Case #2:

Old Behavior: For a file that has world permissions assigned and at least one group permission assigned, group inherited world permissions, and owner inherited world and group permissions.

New Behavior: For a file that has world permissions assigned (regardless of group permission assignment), group inherits world permissions, and owner inherits world permissions.

In this case, owner is granted fewer permissions (because it does not inherit the group permissions) compared to the old behavior. However, a file that had its permissions set by an older server already had its group permissions inherited by owner when the permissions were stored in the file, and so those permissions inherited from group to owner remain for that file unless an administrator changes the file permissions.

Case #3:

Old Behavior: Due to a bug, group and owner never inherited world write permission.

New Behavior: Group and owner inherit world write permission.

In this case, group and owner are granted additional permissions (the world write permission that should have been granted) compared to the old behavior.

Compatibility Change: Due to these modifications in permission inheritance behavior, the `INHERIT_FILE_PERMISSIONS` server configuration option is *no longer supported*.

3.2 Delete Node Reuse List Default Behavior Change

In V11.5 (and FairCom RTG V2.5) and later, deleted index nodes are not reused until after the file is closed. This can cause indexes to grow faster than expected.

A new server keyword `NODE_REUSE_DELAY` allows setting the node reuse time directly. This keyword sets the number of seconds after an empty index node is pruned from the index before it may be reused (default 0). Negative values disable index node reuse until the file has been closed and reopened. Prior to V11.5, this defaulted to 225 and was 0.75 * `NODE_DELAY` (minimum 4).

The deleted nodes are cleaned up when the files are closed, which means that this issue is a concern only if files are held open for long periods of time.

Workaround: A workaround for this problem is to set `NODE_DELAY` to 4 or larger (before V11.5, the default was 300).

Note: This is a change to the Default Behavior. In V11.5.0, `NODE_DELAY` was allowed to be set to zero (which was the default), indicating to prune empty index nodes immediately. Prior to this, the minimum value for NODE_DELAY was 1. This change replaces that behavior with server keyword `NODE_REUSE_DELAY` (default 0). The new `NODE_REUSE_DELAY` keyword defaults to 0.

3.3 FairCom DB API - ctdbNextBatch() no longer returns last record twice when filters are in place

In some cases **ctdbNextBatch()** returned the last record twice when filters were in place. The logic has been modified to correct this:

1. **ctdbNextBatch()** has been modified to handle the case of no error and 0 records returned because no records match the filter criteria. In this situation it returns **BTMT_ERR** so the application gets the error indicating that there are no more records to retrieve.
2. Logic has been added on the server to detect that no records are returned because they were filtered out by record filter criteria. Instead of returning no error, it now returns **BTMT_ERR** as if it were called with no more records to retrieve.

Compatibility Note: Notice that a call to **ctdbNextBatch()** will return **BTMT_ERR** in the cases indicated above. All combinations of client and server with and without the fix should behave properly.

3.4 File compact no longer changes compression type to the database default

When a compressed file was compacted, the compacted file used the database engine's default compression type instead of the original file's compression type. The logic has been corrected so the compacted file now has the same compression attributes as the original file.

3.5 Fix to preserve encryption when compacting files

When enabling encryption for a file while compacting it, if the data or index file had the ctNOENCRYP extended file mode bit set, encryption was not enabled for the file. The file compact now turns off the no-encrypt option when the caller requests that the file is to be encrypted.

3.6 LOKREC() ctCHKLOK mode now returns lock information for lock acquired using co-file

When a c-tree data file was open more than once using the MULTIOPN_SAMUSR_1 or MULTIOPN_SAMUSR_M mode, a call to **LOKREC()** with mode of ctCHKLOK did not return information about a lock that was acquired using a different file number (co-file).

The ctCHKLOK feature has been modified to loop over all co-file numbers for a file that is open using the MULTIOPN_SAMUSR_1 or MULTIOPN_SAMUSR_M mode, checking for the existence of a lock at the specified offset.

3.7 SQL now considers string literal as VARCHAR during parsing

The following SQL statements produced the output shown below:

```
select '[' + f1 +']'  from (select 'abc' as f1
union all
select 'fdsafsda' as f1) as pjoin
```

The resulting output was as follows:

```
[+F1+]
------
[abc     ]
[fdsafsda]
2 records selected
```

Notice the extra spaces on the first row: `[abc      ]`

Using other SQL (such as MS-SQL), the above statement produces different output:

```
[+F1+]
------
[abc]
[fdsafsda]
2 records selected
```

The parser considered the '`abc`' and '`fdsafsda`' string literal as CHAR (fixed-length strings). When doing the UNION, the result definition became the larger of the two fixed-length strings, so the smaller one was padded with spaces.

The logic has been modified so the parser considers the string literal as a VARCHAR, which resolves this issue.

Compatibility Note: This modification is a *behavior change* that may affect the query result. When literals are used as selected values, the result type will be VARCHAR instead of CHAR and results with trailing spaces will not have them anymore. The old behavior can be restored by adding `SQL_OPTION LITERAL_CHAR` in *ctsrvr.cfg*

3.8 DSQL - Corrected bad behavior when getting data with wrong datatype function

Functions for retrieving data from result set (such as **ctsqlGetSmallInt** and **ctsqlGetDate**) could give unexpected results if they were called on a column with a different data type. These two were very likely to crash the client if called on a column with a different data type:

- **ctsqlGetBytes**
- **ctsqlGetBinary**

The logic has been improved to correct this problem:

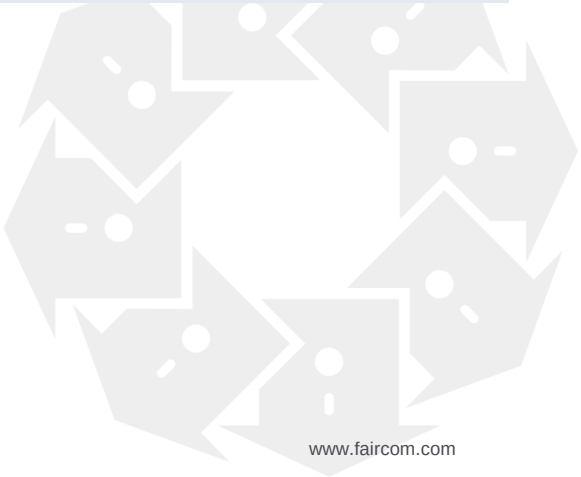
1. Logic has been added to the **ctsqlGet<type>** functions to perform type checking.
   - If the type is not correct and type checking is enabled (see below), **CTSQLRET_INTYPES** is returned.
   - If the type is not correct and type checking is NOT enabled (see below), the data is converted to the requested type. (**ctsqlGetBytes** and **ctsqlGetBinary** do not attempt conversion but simply return the binary content of the field.)
2. A new function, **ctsqlSetTypeCheckOnGet()**, has been added to turn on/off this type checking in case someone want to use the old behavior. The default is to have the checks on. The setting change applies immediately to the connection and all open cursors.
3. The following commands previously returned **CTSQL_BADPARAM** (-20127) when called with a wrong (typically NULL) argument. This error code was not correct, so these functions now return **CTSQLRET_BADARG** (-20068), which is the appropriate error. **Note**: This is a behavior change (it still fails, but with a different error).
   - ctsqlSetSSL
   - ctsqlSetPreserveCursor
   - ctsqlConnect
   - ctsqlDisconnect
   - ctsqlDetach

- ctsqlBegin
- ctsqlCommit
- ctsqlAbort
- ctsqlSetIsolationLevel
- ctsqlSetAutoCommit
- ctsqlSetParameter
- ctsqlSetParameterScale
- ctsqlSetParameterPrecision
- ctsqlSetParameterPrecisionX (this function is not available in all DSQL versions)
- ctsqlPrepareBatch
- ctsqlExecute
- ctsqlClose
- ctsqlDescribe
- ctsqlSetBlob
- ctsqlSetTimeout
- ctsqlCleanPooled
- ctsqlDumpStpRaw
- ctsqlDumpStp
- ctsqlDeployStpRaw
- ctsqlDeployStp
- ctsqlSetScrollableCursor

3.9 DSQL - ctsqlSetParameter() properly handles strings without null terminator

Calls to **ctsqlSetParameter()** failed with -20068 (**CTSQLRET_BADARG**) if they are passed string data without a null terminator. **ctsqlSetParameter()** now appends a terminator if a character-type parameter without a terminator is found.

Compatibility Note: The is a *Behavior Change* because older versions did not check and could transmit the wrong parameter data to the server instead of failing. A revision made calls with a valid length argument but no null terminator fail with **CTSQLRET_BADARG** (-20068); however the PHP PDO driver expects such calls to function properly, so this change was made to add the null terminator automatically.



4. Performance

4.1 Improved Large Transaction Performance

While updating many records in a single transaction, the update rate slowed down over time. This happened even if a table lock was acquired and PreImage memory use was reduced with the `MAX_PREIMAGE_DATA` configuration option. Increasing the `PREIMAGE_HASH_MAX` configuration option improved the situation, however even with that option the update rate slowed over time.

A hash table is used to efficiently search the PreImage space entries containing the updated record images. This hashing has been improved in the following critical areas:

1. A limit of 1048583 hash bins in no longer imposed;
2. An Improved the hash function provides a more even distribution of values over the hash bins.

For maximum performance, we have modified the hash function to use up to 2^31-1 hash bins. We also modified the lock hash function in the same manner.

Prior to this change, dynamic hashing defaulted to a maximum number of hash bins of 128K. `PREIMAGE_HASH_MAX` can raise this limit.

Hint: `PREIMAGE_HASH_MAX` and `LOCK_HASH_MAX` keyword values larger than 1 million can provide additional performance benefits for large transactions.

Default: The default value for the `PREIMAGE_HASH_MAX` configuration option has been increased from 131072 to 1048576 such that a dynamic hash of PreImage space entries is more effective for large transactions without requiring a server administrator to increase this setting.

5. FairCom Server Changes

5.1 Unhandled exception shutting down database engine using Server DLL

When an application uses the Server DLL model, shutting down the database engine by calling **ctThrdTerm()** may cause an unhandled exception in an internal thread. The logic has been modified to correct this.

5.2 Fixed FairCom Server unhandled exception when using preimage swap file and log encryption

When using the preimage swap feature with log encryption enabled, FairCom Server sometimes terminated with an unhandled exception. The logic has been modified to correct the problem.

5.3 c-tree Server rare abnormal termination with Dynamic Dump !PROTECT option fixed

In certain rare situations when the Dynamic Dump `!PROTECT` option was used, c-tree Server could terminate abnormally if the delete node thread's node pruning operation was in progress when the Dynamic Dump was backing up that same index.

- On Linux, the symptom is an abnormal c-tree Server shutdown occurring due to an error in an OS reader/writer lock request that had failed with an **EAGAIN** error.
- On Windows, an exception could occur.

The logic has been modified to correctly handle this situation.

Note: It was determined an internal **pthread** reader/writer lock failed very unexpectedly with a system **EAGAIN** error, indicating the limit of concurrent readers has been reached. Now, when this internal error is encountered, the thread waits and retries up to 5 times before triggering a fatal error.

5.4 Rare unexpected shutdown with error 8757 fixed

An unexpected rare server shutdown could be experienced with error **8757**. *CTSTATUS* contained the following message:

```
Unexpected internal c-tree(R) error #8757 (uerr_cod=0).
```

Similar fatal errors **8758** or **8760** are possible in similar scenarios. The logic has been modified to prevent this shutdown.

5.5 Infinite loop when IICT updates index member more than once

An infinite loop was seen while committing an Immediate Independent Commit Transaction (IICT) when an index member that was not updated in the outer transaction was updated more than once in the IICT. The logic has been updated to prevent this problem.

5.6 Index Node Split Error

In extremely rare situations, after a parent-child branch error is detected in the tree structure of an index, there was a remote chance that FairCom Server could terminate. Messages in *CTSTATUS.FCS* were similar to the following:

```
 - User# 00312 Could not adjust path: 617
 - User# 00312 Parent-Child branch error: 1
 - User# 00312 Dumped stack for server process 1638724, log=1, loc=81, rc=0
 - User# 00312 O312 M18 L81 F1 P0x (recur #1) (uerr_cod=617)
```

The logic has been modified to correct this problem, eliminating the need for the FairCom Server to shut down.

This is an extremely rare event because it requires an index error, such as the parent-child branch error, followed by a failure to adjust the path from the newest root to the node. These are both extremely rare events, and both happening at the same time is an extremely remote possibility.

5.7 Fixed unexpected c-tree core activity while server is in Quiet state

In certain situations, *CTSTATUS* showed **WRITE_ERR** messages made during a server Quiesce by a user whose transaction was aborted by **ctQuiet()**:

```
User# 00027 ctQUIET: blocking call with action: 641a2x
User# 00027 Transaction aborted at ctQTaborttran for user# 24: 817
User# 00027 ctQUIET: end of blocking call
User# 00024 WRITE_ERR: I0000001.FCS at 0:0x sysiocod=6 bufsiz=128 bytes written=0[0] ioLoc=0: 37
```

The logic has been modified to ensure these errors do not occur.

5.8 Startup/Recovery no longer fails (error 66, RCHK_ERR) in rare circumstances involving large forward roll

Startup/Recovery failed with error 66 (**RCHK_ERR**) following a large forward roll. In this case, the checkpoint created by the forward roll contained a large redo list, making the size of the checkpoint entry nearly 300MB. Recovery was set to fail with **RCHK_ERR** if a checkpoint entry exceeded 16MB. The checkpoint size limit at recovery time has been increased to 2GB.

5.9 Unlock failure while closing file no longer causes infinite loop

The FairCom Server could hang or become unresponsive (due to an infinite loop) if an unlock call failed while closing a file. The logic has been modified to eliminate this problem.

5.10 Fixed c-tree Server unhandled exception when using background flush

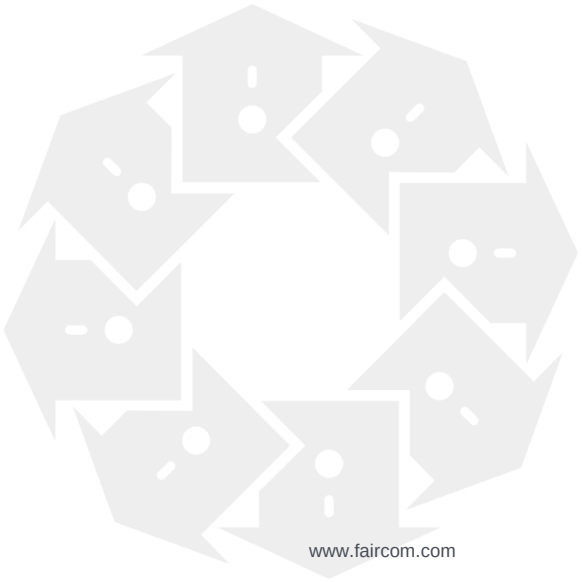
In an extremely rare situation when using c-tree Server's background flush feature, the server could terminate with an unhandled exception in the internal background cache flush logic.

Workaround: Disable the background flush buckets by adding these options to *ctsrvr.cfg*:

```
NONTRAN_DATA_FLUSH_BUCKETS 0
NONTRAN_INDEX_FLUSH_BUCKETS 0
TRAN_DATA_FLUSH_BUCKETS 0
TRAN_INDEX_FLUSH_BUCKETS 0
```

The buckets are optional (although they provide a histogram of the distribution of updates, the flush threads can operate normally without them).

The logic has been modified to eliminate this problem.



6. FairCom DB SQL Server

6.1 SQL - OVERLAY function now compliant with SQL:1999 standard, LPAD/RPAD do not overflow

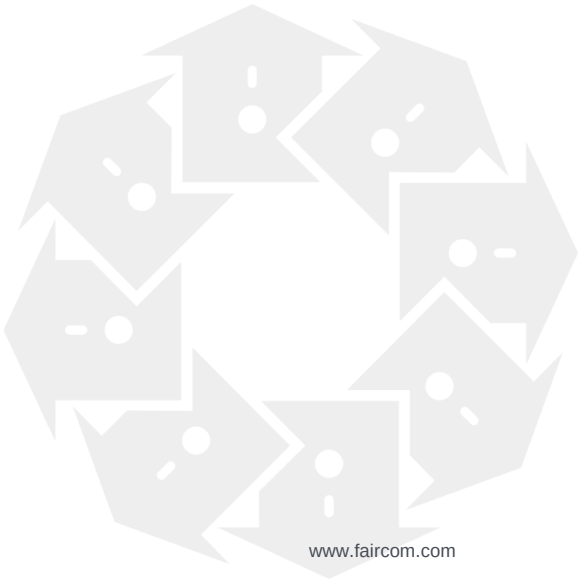
The SQL:1999 standard on the OVERLAY function specifies:

```
OVERLAY (<character value expression>
PLACING <character value expression>
FROM <start position>
[ FOR <string length> ] )
a) Let CV be the first <character value expression>, let SP be the <start position>, and let RS
be the second <character value expression>.
b) If <string length> is specified, then let SL be <string length>; otherwise, let SL be CHAR_
LENGTH(RS).
c) The <character overlay function> is equivalent to:
SUBSTRING ( CV FROM 1 FOR SP - 1 )
|| RS
|| SUBSTRING ( CV FROM SP + SL )
```

Our implementation was not compliant because, when `<string length>` was not specified, it inserted the second expression into the first one at the specified position without overwriting the content, which is incorrect.

The LPAD and RPAD functions did not generate overflow error if the length passed in is larger than the maximum field length, now they do.

Behavior Change: This modification is a change in behavior.



7. Security

7.1 Ensure recoverability if server terminates while changing master encryption key

The changing of the master encryption key involves multiple steps. If an error occurs, the work is undone, but if the server terminates we did not ensure that the changes were all undone. This could render the server unable to start or unable to decrypt files.

If FairCom Server terminates while the master encryption key is being changed, the encrypted files (data and index files and transaction logs) might have been left in a state that is inconsistent with the current master key, causing them to fail to open with errors **606**, **607**, or **66** on the next restart of FairCom Server.

A write-ahead recovery log has been implemented for the master encryption key change. The log file, named *CMPRECOV.FCS*, is created in the same directory as the transaction logs. It records the change we are about to make to a file before we apply the change. When FairCom Server starts up, if this log file exists FairCom Server uses the information in this recovery log to undo all the changes to the files so the data, index, and transaction log files are all using the original master encryption key. This log also allows c-tree to undo the changes to non-transaction files, which are not recorded in c-tree's transaction logs.

7.2 Fixed error reading transaction logs using advanced encryption

When using transaction logs that were encrypted with advanced encryption, a transaction log reader (such as the internal server thread that processes Full-Text Index, deferred key, and record update callback log entries) might fail to read the transaction logs. In this situation, a message such as the following was logged to *CTSTATUS.FCS* and the internal thread entered a paused state:

```
DeferredIndexer: Failed to read next change from transaction log: 76
```

The error code may vary; for example, error **75** was also observed in this situation. The logic has been modified to prevent this error from occurring.

8. Data Integrity

8.1 Dynamic Dump and Backup/Restore

Dump restore immediately after dump backup may skip rolling files back to the dump start time

If a dynamic dump restore happened within one second of when the dynamic dump backup started, the dump restore could skip rolling back the changes to the files that occurred during the

dynamic dump, causing the files to contain unexpected modifications. The logic has been modified to correct this.

c-tree Server no longer logs immediate restore error message on Unix even when immediate restore is successful

c-tree Server logged the following message to CTSTATUS.FCS on the Unix platform even when an immediate dump restore succeeded:

```
- User# 00021  DD: !IMMEDIATE_RESTORE dump restore execution failed: 60
```

This was also seen on Mac OS X (error code 60) and AIX (error code 10).

When reading the output of the ctrdmp process on Unix, the logic considered a zero-byte read to be an error instead of simply indicating the end of the file. The logic that reads ctrdmp output on Unix has been modified so that a zero-byte read is considered a normal end of file situation.

The server has been modified to properly not detect ctrdmp returning an error on Unix and Windows.

Affected Components: c-tree Server

Immediate restore - Fixed incorrect parsing of local directory name from script name command-line option

When a dynamic dump script used the immediate restore feature and specified a relative path to the dump stream file, the immediate restore could fail with the error message "Could not set current directory to <directory>." (Use the `!IMMEDIATE_RESTORE_LOGFILE` dump script option to log **ctrdmp** immediate restore output to the specified log file.)

This was caused by improperly parsing the server's local directory from the script name specified in the command line arguments, causing the wrong directory name to be used for the dump restore.

The logic has been corrected so the server now passes a new command line option, *-L <local_directory>*, to the **ctrdmp** utility.

Workaround: Use a full path instead for a relative path for the `!DUMP` option in the dump script.

Compatibility Note: A server that uses the *-L* option requires a **ctrdmp** utility that supports that option. Otherwise, the immediate restore fails.

Affected Components: c-tree Server, ctrdmp utility

Dynamic Dump fixes

The following fixes have been made to Dynamic Dump:

- Unhandled exception could be seen after a Dynamic Dump failed. This issue has been resolved.

- An unexpected debug error message could be seen in selected cases when the dump script used wildcards and some wildcard specifications had no matching files. This problem has been fixed.
- When the options `PREIMAGE_DUMP YES` and `PERMIT_NONTRAN_DUMP NO` are specified in *ctsrvr.cfg*, ctPREIMG files are expected to be switched to ctTRNLOG files for the duration of the Dynamic Dump and non-transaction files are not included in the Dynamic Dump. However, a bug caused some PREIMG files to be mistakenly excluded from the dump and some non-transaction files to be included. This error has been corrected.
- In situations when a file could not be read due to a system error, the **ctdump** utility with `!BLOCK_RETURN` in the backup script reported a successful backup. *CTSTATUS.FCS* had a log entry indicating non-zero sysiocod:
  ```
  - User# 00019 getfile:ictio81: retsiz=0 recbyt=0000-004b0000x
  lstbyt=0000-013998fcx CHUNK=65536 sysiocod=23 :: 0xff fill: 443
  ```
  The logic has been modified so that, when reading a file for dynamic dump and a system error occurs, it now reports this as a **READ_ERR** (36) and terminates the dynamic dump.
- Dynamic Dump Restore failed with error **934** if Immediate Independent Commit Transaction (IICT) activity occurred during a dynamic dump backup. The logic has been modified to correct this.

8.2 Automatic Recovery

Fixed automatic recovery error 12 if server terminated after PUTHDR() ctTIMEIDhdr

Automatic recovery error **12** was seen when FairCom Server terminated abnormally after **PUTHDR()** was used to change the *timeid* file ID header field of a ctTRNLOG file. This situation has been addressed.

Automatic recovery - Improved handling of duplicate file IDs and transaction-dependent operations

In cases involving transaction-dependent file creates and renames, a duplicate file ID is caused by a system copy of a file, which could cause problems during automatic recovery. Symptoms included recovery failing with internal error **8777**, or record counts being incorrect after automatic recovery, or possibly even an unhandled exception in very obscure situations. If automatic recovery opened the renamed file and the original file, their file IDs could conflict, causing internal error **8777**. Or the redo of the create could cause the record count in the data file header to be reset to zero, or adjusted for operations on the copied or renamed file.

Automatic recovery has been modified to address this situation.

Automatic recovery error L64, 1120 fixed

Automatic recovery sometimes failed with error **L64** when the file was deleted and re-created after the last checkpoint in transaction log. Or it could report error **1120** during the transaction undo phase. The logic has been modified to prevent this.

In case of L64 recovery failure, log a message to CTSTATUS.FCS and return an error rather than exiting the process

In an unusual situation (an L64 recovery failure), automatic recovery could fail due to an error mapping a transaction file number to a c-tree file number, causing the process to be exited.

The logic has been modified to return the error to the caller instead of exiting. Now the c-tree initialization function will fail with error **1120** in this situation. This applies to both standalone TRANPROC and server models. Messages will be written to *CTSTATUS.FCS*. Here are example messages logged by c-tree Server in this situation:

```
Wed Jan 31 14:29:52 2018
 - User# 00001  L64: tfil:10 lgn:3 updflg:0 curval:-4: 1120
Wed Jan 31 14:30:07 2018
 - User# 00001  Automatic recovery terminated with error: 1120
Wed Jan 31 14:30:09 2018
 - User# 00001  Could not initialize server. Error: 1120
Wed Jan 31 14:30:11 2018
 - User# 00001  Dumped stack for server process 65912, log=1, loc=74, rc=0
Wed Jan 31 14:30:11 2018
 - User# 00001  O1 M0 L74 F1120 P0x (recur #1) (uerr_cod=1120)
```

Automatic recovery failure with error 413 fixed

Automatic recovery failed with error **413** if it tried to redo the create of a superfile member and the host did not exist, even if `SKIP_MISSING_FILES YES` was used. *CTSTATUS.FCS* showed messages with error **413** and indicated that recovery terminated due to that error.

When using `SKIP_MISSING_FILES YES` a missing superfile host is not expected to cause automatic recovery to fail.

The logic has been modified so that (when using `SKIP_MISSING_FILES YES`), recovery continues if opening a superfile member fails because the superfile host can't be opened. Now when this situation occurs, if `DIAGNOSTICS TRAN_RECOVERY` is in effect, we log the following message to RECOVERY.FCS:

```
<skip redo create of superfile member due to missing superfile host: SUPERFILE_MEMBER_NAME>
```

where SUPERFILE_MEMBER_NAME is the name of the superfile member whose create was being redone.

Automatic recovery may fail with error 8776 after delete and rename of superfile member

Automatic recovery may fail with internal error **8776** after an abnormal server shutdown following the deletion and rename of a superfile member. This has been observed when FairCom DB SQL Server terminated after it had converted a SQL database dictionary superfile from an old version to the current version at server startup.

The following messages were seen in *CTSTATUS.FCS*:

```
- User# 00001 Unique File ID error 13,0]: op=1 fid=19282491 sid=89030210x tid=5d08f288x: 2
- User# 00001 ./live.dbs/SQL_SYS/live.fdd!Z956A3E8.dat: 2
- User# 00001 Unexpected internal c-tree(R) error #8776 (uerr_cod=2)
```

Improvements to automatic recovery's detection and handling of duplicate file IDs caused a change in behavior when redoing a committed transaction-dependent rename of a superfile member.

Workaround: Add `RECOVERY_CHECK_DUPFID NO` to *ctsrvr.cfg* and re-run automatic recovery.

The logic has been modified to correct this problem.

Affected Components: This modification affects c-tree's automatic recovery feature, in standalone and server models. This means that the bug fix applies to code in the c-tree single-user TRANPROC library and to the c-tree Server database engine.

Improve Automatic Recovery detection of a mismatched file caused by copying a file outside the server

Before running automatic recovery, if a file is copied to the server's data directory, overwriting another file having a different file name, in some cases automatic recovery doesn't detect that the file is not the proper file. Two changes were made to automatic recovery to handle cases that involved the rename of a file:

1. The recovery step that opens the file by its renamed filename ignored a file ID mismatch. Now we check if the file ID mismatch is explained by a subsequent transaction-dependent operation such as a file create or a rename of another file name to that file name with that file ID. If so, we allow recovery to continue; if not, we log the message "Detected file ID mismatch *file*:" and the file name to *CTSTATUS.FCS* and fail recovery with a file ID mismatch error, **499**.
2. If an undo of a rename fails, recovery fails (previously, recovery proceeded in this situation).

Improved Detection of Logically Equivalent Filenames during Automatic Recovery

Opening a file using a different, but logically equivalent path specification could prevent automatic recovery from determining that the file was part of a transaction-dependent rename operation. This could happen with two different physical files that have the same file ID. The filename check for transaction-dependent file operations has been improved so the logically equivalent path specifications are determined to be for the same file.

8.3 Transaction Processing

Error 160 after switching an index's transaction mode fixed

Error **160** was seen after switching a transaction-controlled index into non-transaction mode. The error was caused by transaction-mark cleanup optimization skipping the clean-up of some marks. The logic has been modified to correct this.

Workaround: Add the following options to *ctsrvr.cfg* and restart FairCom Server:

```
COMPATIBILITY NO_MYMARKS
COMPATIBILITY NO_NXTMARKS
COMPATIBILITY NO_UPDMARKS
```

Note that these configuration options will only prevent future occurrences of the error. Any indexes already experiencing errors will need to be rebuilt.

Open of transaction-controlled mirrored file on Unix no longer fails with error 549

When the `LOCAL_DIRECTORY` option was used on a Unix system, opening a transaction-controlled mirrored file could fail with error **549**. Messages in *CTSTATUS.FCS*:

```
 - User# 00017  Reassigning mirror file ID ...
 - User# 00017  qafilnam_mir.dat
 - User# 00017  The primary in the following pair appears to be the most current: 549
 - User# 00017  qafilnam.dat|qafilnam_mir.dat
 - User# 00017  qafilnam.dat|qafilnam_mir.dat:549
```

The logic has been modified to correctly handle this situation.

PREIMAGE_DUMP and ctDEFERBEG interaction no longer causing long dump delay

When running a dynamic dump with `PREIMAGE_DUMP YES` in *ctsrvr.cfg*, the dump may ignore the `!DELAY` setting and block new transactions until all open transactions complete, including open SQL read-only transactions. The logic has been modified to correct this.

Workaround: Disable `PREIMAGE_DUMP YES` in *ctsrvr.cfg* and use an alternate backup approach to ensure a consistent backup (e.g., shut down the c-tree Server and do a file copy, invoke a c-tree **quiesce** operation using the **ctquiet** utility, etc.).

Rare KDUP_ERR resolved

Adding a key to a transaction-controlled file could fail with a duplicate key error (**KDUP_ERR**, error code 2) when FairCom Server's `NXTMARKS` optimization was enabled (which is on by default). This error requires the following specific sequence of events to occur:

1. Start a transaction. Use an index lookup function such as **EQLKEY** (or **EQLREC** if using ISAM level) to read the last key in a leaf node. Use **NXTKEY** (**NXTREC**) to read the next key, which is the first key in the successor leaf node. Delete this key and add it with a different offset. Abort the transaction.

2. Repeat step 1. This time the key add fails with a KDUP_ERR.

The **KDUP_ERR** can occur for this specific sequence of events without any other intervening accesses to the leaf node. However, if other accesses to the leaf node occur before the new key is added, the error may not occur.

The logic has been modified to eliminate this error.

Workaround: Add `COMPATIBILITY NO_NXTMARKS` to *ctsrvr.cfg* and restart FairCom Server.

Affected Versions: V10.0.0 through V11.6 lines prior to November 2018

Affected Components: FairCom Server

Fixed transaction log read errors when using delayed durability and encrypted transaction logs

When using delayed durability and encrypted transaction logs, there was an issue in the server code causing wrong information to be written in the transaction log files. This affected automatic recovery and features requiring the reading of the logs such as replication and deferred indexing.

Log reads performed by an internal thread or a replication log reader sometimes failed with a checksum error or with errors such as **75** or **76**. In this situation, error messages such as the following were logged to *CTSTATUS.FCS*:

```
- User# 00011    DeferredIndexer: ERR: Failed to read next change from transaction log: 75
- User# 00011    DeferredIndexer: ERR: Log read position: log=1, offset=2479283
- User# 00011    ctrepl: Checksum failure at code location 1: 36
- User# 00011    DeferredIndexer: ERR: Failed to read next change from transaction log: 36
- User# 00011    DeferredIndexer: ERR: Log read position: log=1, offset=26017799
- User# 00011    ctrepl: Checksum failure at code location 1: 36
```

The logic has been modified to eliminate this problem.

For the sake of additional safety, the transaction log flush function now checks if its write to the transaction log failed, and if so it shuts down the database engine with internal error code **8523**. This is extremely unlikely to happen and the check is included only for completeness. **Note**: This is a behavior change.

Affected Versions: V10.5.0 and later (only if delayed durability and log encryption are both used).

Affected Components: c-tree Server.

Delayed durability log flush no longer causes rare invalid transaction log entry

When using the delayed durability feature, the following symptoms were sometimes observed:

- Reading transaction logs may fail with an error such as error **75** or **634**.
- Automatic recovery may fail with an error such as error **75** or **634**.
- FairCom Server may terminate with internal error

A problem could cause incorrect data, such as 0xff fill bytes instead of the proper log entry data, to be written to the transaction log.

The logic has been modified to eliminate this behavior.

When a transaction deletes a key and adds it with a different offset then aborts, subsequent add of the key no longer fails with KDUP_ERR

Inserting a key into a transaction-controlled index could unexpectedly fail with a duplicate key error when the NXTMARKS and UPDMARKS optimizations were enabled if the key was previously deleted and added in a transaction and then that transaction was aborted. The logic has been modified to correct this.

Workaround: Add `COMPATIBILITY NO_NXTMARKS` or `COMPATIBILITY NO_UPDMARKS` to *ctsrvr.cfg* and restart c-tree Server.

Using c-tree library without transaction support to rebuild or compact a file no longer removes ctTRANMODE / ctPIMGMODE mode

When using a c-tree library compiled without transaction support to rebuild or compact an index that has the ctTRANMODE or ctPIMGMODE attribute set, the new index file did not have that attribute set. The rebuild and compact logic has been modified to ensure it creates the index file with the same extended create mode attributes as the original file.

TLOG_ERR after turning off transaction control for a file fixed

After turning off ctTRNLOG transaction control for a file, a subsequent modification to the file such as a record add, delete, or update can produce a **TLOG_ERR** message in *CTSTATUS.FCS*, followed by other errors such as transaction log write error **541**, or read or write errors on transaction-controlled files. This can cause c-tree Server to terminate because the transaction logging subsystem cannot proceed. This error is a timing related error and may not occur in every situation.

The message "`ctwrtlog: log end error: 73`" in *CTSTATUS.FCS* indicates that this error situation has occurred. After the message occurs, the server will not be able to perform any more transaction operations and is very likely to terminate.

The logic has been modified to prevent this from occurring.

Compatibility Note: The following restriction was added to **PUTFIL()**: Changing a file from ctTRNLOG to ctPREIMG mode is not allowed if the file has been updated in a transaction that is still active when **PUTFIL()** is called. In this case, the error code **XTRN_ERR** is returned. This restriction is designed to avoid possible unexpected behavior. **PUTFIL()** already returned **XTRN_ERR** when attempting to turn off transaction control for a file that has been updated in a transaction that is still active.

This error cannot happen in the single-user TRANPROC model because it doesn't support the deferred OPNTRAN optimization.

TransactionHistory() fixes

The following bugs in the **TransactionHistory()** function have been fixed:

- Forward search did not find any entries for the file even though they exist in the logs.
- The function returned success but did not return data to the client. As a result, the input buffer contained whatever value it contained on input, which could be an uninitialized value.
- Memory leaks.
- Backward search might not find entries that preceded an automatic recovery.
- Backward or forward search might not find entries if a file's file ID was reassigned.

To use transaction history, the following option must be added to *ctsrvr.cfg* **before the log entries** that transaction history is going to scan are written to the transaction logs:

```
COMPATIBILITY TRAN_HISTORY_LOG_ENTRIES
```

If transaction history is called without this option enabled, it returns a not supported error (**454**) and logs the following message to *CTSTATUS.FCS*:

```
“Transaction history requires COMPATIBILITY TRAN_HISTORY_LOG_ENTRIES in ctsrvr.cfg, and the transaction log entries must be written with this option in effect.”
```

New options have been added to the transaction history example program, **trnhis**, to allow specifying backward or forward scan and optional starting transaction log number. If not specified, the log number for a backward scan defaults to the server's current transaction log, and for a forward scan defaults to the server's oldest active transaction log.

Limitation: In single-user tranproc mode on Windows, the history function fails with error **611** if run on a transaction log that has been created and not yet closed and reopened. If this error occurs, restart the single-user process so the history function can access the transaction log.

9. Relational FairCom DB SQL

9.1 SQL

SQL now considers string literal as VARCHAR during parsing

The following SQL statements produced the output shown below:

```
select '[' + f1 +']'  from (select 'abc' as f1
union all
select 'fdsafsda' as f1) as pjoin
```

The resulting output was as follows:

```
[+F1+]
------
[abc     ]
[fdsafsda]
2 records selected
```

Notice the extra spaces on the first row: `[abc      ]`

Using other SQL (such as MS-SQL), the above statement produces different output:

```
[+F1+]
------
[abc]
[fdsafsda]
2 records selected
```

The parser considered the '`abc`' and '`fdsafsda`' string literal as CHAR (fixed-length strings). When doing the UNION, the result definition became the larger of the two fixed-length strings, so the smaller one was padded with spaces.

The logic has been modified so the parser considers the string literal as a VARCHAR, which resolves this issue.

Compatibility Note: This modification is a *behavior change* that may affect the query result. When literals are used as selected values, the result type will be VARCHAR instead of CHAR and results with trailing spaces will not have them anymore. The old behavior can be restored by adding `SQL_OPTION LITERAL_CHAR` in *ctsrvr.cfg*

SQL - Failed update statement now returns correct error

An issue was discovered with an update statement on a table with a unique or a primary key that caused a duplicate key error. The statement should fail with the following:

```
Error: -17002 Description: CT - Key value already exists in index...
```

Either of the following incorrect results could occur:

- it failed with a different error
- the transaction could be placed in a bad state, causing further operations to fail with:

```
error(-17094): CT - Pending error - cannot save or commit tran
```

The issue does not affect the data because it fails. However, the error code was not properly propagated and the transaction may not be restored to the last good savepoint. The logic has been modified to correct this.

SQL - Enforce record size checking

FairCom Database Engine has been updated to enforce its 64KB per record size limit. Prior to this update, a record could exceed this limit, possibly corrupting the record's content. Similar problems also were found when a record must be swapped to disk with unpredictable consequences. This modification corrects these problems with these new features:

- Checks when initializing the record descriptor to verify that the record fits within the maximum record size which is calculated based on the free space on an empty page. If not, c-tree fails with an error: **-16324**
- Checks when inserting into a new page to verify that the record fits, if not, c-tree fails with an internal error: **-16304**

SQL - LIKE now returns correct result in Unicode server

When using an ODBC driver with a Unicode server, it failed to retrieve the table list. The problem was not limited to ODBC; it affected our implementation of the LIKE statement. The logic has been modified to better identify these cases.

SQL - LONG field handle validation

FairCom DB JDBC returned a **CTDBRET_NOROWID** error from a SQL INSERT operation on a table that did not and should not have an internal ROWID column. This was caused by a field handle being corrupted. We have added additional validation for field handles and will now return **-17692** in this instance to signify the handle failed a validity check.

ctdbAlterTable in a CTSESSION_SQL session no longer removes synonyms and table granted permissions

When the current session was a CTSESSION_SQL (i.e. automatic handling of SQL system table on table create/alter and drop), **ctdbAlterTable(...)** removed granted permissions and synonyms from the system table. The logic has been enhanced to eliminate this problem.

A similar problem occurs in FairCom RTG when running **sqlize** on tables already linked into SQL. That problem has not been fixed in this modification.

Note: This modification is a Compatibility Change.

9.2 SQL Stored Procedures

SQL Stored Procedures - Close cursors that were left open

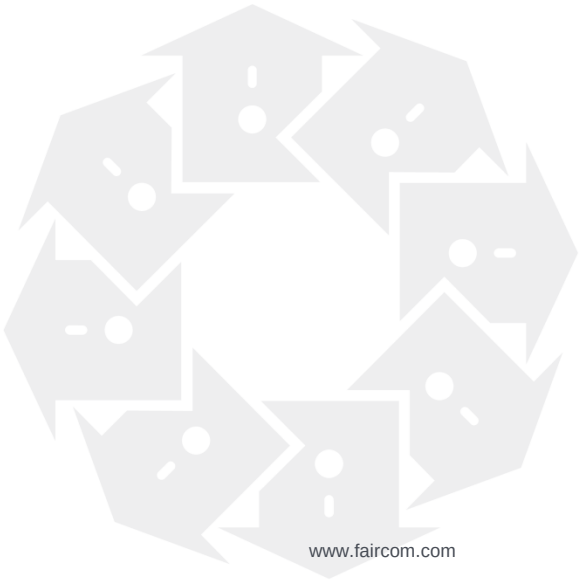
In a rare situation involving a complex set of stored procedures, cursors could leak. In c-tree stored procedures, cursors need to be closed explicitly; if this is not done, the cursor is leaked.

Logic has been added to the function that starts stored procedures to identify cursors that are opened and not closed within a stored procedure and close any that are found. When Java debugging is turned on in TPESQLDBG, a message is logged in *sqlserver.log* to help in identifying the stored procedure and the statement that leaked the cursors.

9.3 Java Hibernate driver now correctly executes SQL statement with SKIP clause

In prior versions, the Java Hibernate Driver did not properly execute SQL statements containing the SKIP clause. This has been fixed by changing the Hibernate configuration setting `supportsLimitOffset=true`.

Affected Components: Hibernate Dialect (CtreeDialect.java)



10. Core Engine

10.1 Communications Layer

Improved shared memory performance on system with one CPU core

An earlier performance optimization, called "shared memory spin," could slow shared memory performance on a system that has only one CPU core. The logic has been enhanced to detect a system that has only one CPU core and then does not enable the shared memory spin.

Slow connection or shared memory error fixed in specific situation

After c-tree Server encountered an unexpected error in the shared memory protocol and recovered from the error, clients could take a long time to connect and they could fail to connect with shared memory, connecting with TCP/IP instead. Messages in *CTSTATUS.FCS* in this situation:

```
User# 00020  FSHAREMM: Wait on logon semaphore failed: 43
 - User# 00020  FSHAREMM: The SEM_LGON semaphore is invalid. Reinitializing shared memory support.
 - User# 00020  FSHAREMM: Successfully reinitialized shared memory support.
 - User# 00020  FSHAREMM: Read of socket/client pipe failed: 0
 - User# 00020  FSHAREMM: Failed to write to server pipe: 32
```

The logic has been modified to correct this situation.

Client no longer loses connection to server after a failed CompactIFile() call when using TCP/IP protocol

After a failed call to **CompactIFile()**, the next call to the server might return a communication error such as error code **128** when it was using the TCP/IP communication protocol. An error could trigger a call to delete a temporary file, which caused information to be written to the communication buffer unexpectedly, causing a communication error. The logic has been modified to correctly close the temporary file without causing a communication error.

Affected Components: c-tree Server

10.2 Indexing

Rare index corruption following recovery fixed

A very rarely encountered problem involving index pruning could lead to a corrupted index following a database automatic recovery:

```
Error 14 at open.  CTSTATUS.FCS has messages:
User# 00002 ERROR: Invalid node links
Tue Oct 30 02:13:55 2018
User# 00002 MyFile.idx [1]: 527
```

The requirements to see this issue are an index created with the performance optimization LOGIDX (which speeds up automatic recovery and is on by default starting in V10.3 and later) and the index must have at least 3 levels or more in the index tree, and requires deleting all of the key values in an index node, and its parent node. Then if the c-tree Server process is improperly closed, and automatic recovery occurs, this issue might occur.

The logic has been modified to correct this.

Deferred index thread shutdown delay fixed

If FairCom Server shuts down while the deferred indexing thread for TRNLOG files is reading the transaction logs, the thread might log the following message to *CTSTATUS.FCS* and enter a paused state:

```
DeferredIndexer: ERR: Failed to read next change from transaction log: 150
```

This can delay server shutdown and might cause the server to skip writing its final checkpoint because the thread doesn't shut down. The logic has been modified to correct this.

Deferred Index thread now retries in case of log read error

Some cases have been observed in which the deferred indexing thread for TRNLOG data files encounters an error (such as error **75** or **76**) reading the transaction log. In this situation, the thread enters a paused state.

The thread's operation has been modified so that after it pauses, it immediately resumes operation, which may allow it to recover from the unexpected error situation. The thread tracks the log position at which the last log read error occurred, and it only retries up to 3 times at the same log position. (If an error occurs at a different log position, the retry counter is reset to zero so that an error at that location can be retried up to 3 times.)

The thread also now logs the log number and offset at which the log read error occurred. Sample messages from *CTSTATUS.FCS*:

```
 - User# 00011  DeferredIndexer: ERR: Failed to read next change from transaction log: 75
 - User# 00011  DeferredIndexer: ERR: Log read position: log=2, offset=262228
```

Index create or rebuild now requires much less time when using large SORT_MEMORY values

Adding or rebuilding an index may take an extremely long time when using a large `SORT_MEMORY` setting. The logic has been modified to select the most optimal sorting algorithm to avoid the worst-case performance.

Table locks no longer prevent reuse of empty index nodes

Table locks were unexpectedly preventing the reuse of empty index nodes, causing indexes to consume more disk space than expected. Rebuilding
affected indexes will recover the space. Upgrading to V11.5.1 or newer will likely automatically prune this space over time. The logic has been modified to correct this and prevent this issue going forward.

10.3 File Management

Superfile member open no longer fails with POPN_ERR

When c-tree Server's file open optimizations were enabled, opening a c-tree file using an "aliased" name (a symbolic or hard link or, on Windows, a network share UNC name) could fail with error **920** (on Windows) or **463** (on Unix). The logic has been enhanced to properly handle this situation: The system now maintains a list of the system file IDs of files it opens. It uses this list to determine if a file is already open (or pending open) under a different alias.

Affected Components: c-tree Server, ctquiet utility

New Configuration Options

To prevent possible errors if two connections open the same physical file using two different aliases at the same time when the c-tree Server's file open optimization enhancement is effective, the c-tree Server now maintains a list of the system file IDs of the files that it opens. This list makes it possible to determine that a file is already open, or pending open, under a different alias. The following keywords control this new system file ID list:

`SYSTEM_FILE_ID_LIST YES | NO` - Defaults to YES. Enables the system file ID list. Can be turned on or off at runtime when the server is in a quiesced state.

`DIAGNOSTICS SYSTEM_FILE_ID_LIST` - Enables logging of adds/deletes to the system file ID list to the file *SYSIDHASH.FCS* in the server's LOCAL_DIRECTORY directory. Can be turned on or off at runtime.

Note: The file open optimizations (`OPTIMIZE_FILE_OPEN` configuration option) can also be turned off at runtime when the server is in a quiesced state.

To allow an administrator to change server configuration options that can only be changed when the server is in a quiesced state, we added an option to the **ctquiet** utility. The *-m* option can be specified one or more times in a call to **ctquiet**. This option quiesces the server, changes the specified configuration options, and resumes normal server activity. This process avoids additional calls to the Server, which could increase the risk of ending up with an abandoned

quiesced server (meaning the process that quiesced the server has failed, leaving the FairCom Server in a quiesced (quiet) state).

See the **ctquiet** documentation for complete utility options

Example:

```
C:\> ctquiet -p ADMIN -m "optimize_file_open no" -m "system_file_id_list no"
```

Note: The system file ID list requires the file open optimization, so if a request is made to turn on the system file ID list, we also turn on the file open optimization, and if a request is made to turn off the file open optimization, we turn off the system file ID list.

ctdbGetRecordKeyPos no longer causes ctdbNextRecord unexpected behavior on partitioned files

Unexpected behavior was seen on partitioned files:

- When **ctdbGetRecordKeyPos()** was called during an index scan on a partitioned file, the scan never terminated.
- Independent index scans with different record handles (or ISAM contexts) using a partition index could return an incorrect number of search results (too few or too many).

The logic has been modified to correct this behavior.

Note: ISAM applications using a partition index must be aware that a low-level key search on the same index could modify the internal state of their current ISAM context. This does not happen with non-partition files. To avoid unexpected behavior, such low level operations must occur within a separate ISAM context.

Invalidate variable-length data file space management index in case of error

Error **14** was seen when opening a variable-length data file after seeing the following message in *CTSTATUS.FCS*:

```
Could not update space management index during end-of-transaction clean up: 527
```

Logic has been modified to better handle this situation: When an index integrity error is detected, the space management index is invalidated, which permits the data in the file to be accessed. All deleted nodes that were referenced by the space management index are orphaned (this space can be reclaimed by compacting the data file). When the space management index is invalidated, one of the following messages is logged, followed by the data file name:

```
Invalidating space management index due to unexpected deleted node during iaddkey in file...
Invalidating space management index due to unexpected deleted node during ctdelkey in file...
```

To reclaim the delete space in this case, an index rebuild will be needed using **RebuildIFIL()** or the **ctrebuildif.exe** utility.

ctCopyFile function now opens files in exclusive mode to ensure a clean copy is made

In V11.6 and later, the file copy function, **ctCopyFile()**, now opens the file in exclusive mode, as documented. Prior to this modification, it opened the file in shared mode, which allowed updates to be made while the file was being copied, causing the copied files to fail to open with error **14**.

Behavior Change: This change makes the function's behavior match its documented behavior, which introduces a change in behavior in the field. Previously it was possible to copy a file that was open by another connection and if the file was not updated, the copy of the file would be in good shape. But now the file copy will return an error in this situation.

The following new option for **ctCopyFile()** provides a way to use the old behavior of opening the file in shared mode. This option indicates the server should try to open the files to copy in shared mode if the files can't be opened in exclusive mode.

To use this option, OR the *ctCFallowSharedOpen* bit into the *cpoptions* field of the CPYPRM structure passed to **ctCopyFile()**.

CHECKLOCK_FILE configuration option now supports wildcard characters

The `CHECKLOCK_FILE` server configuration option now supports wildcard characters.

File open optimization no longer causes errors opening or deleting files

When c-tree Server was run with the file open and close optimization enabled, if two connections open the same file at the same time using different path specifications, one of the open calls could fail with the following symptoms:

On Unix - File open may fail with error 463 (**UQID_ERR**, file uniqueness error) and unique file ID error messages such as the following may be logged to CTSTATUS.FCS:

```
- User# 00023  Unique File ID error [7,0]: op=1 fid=1219 sid=39000011x tid=5d9b8dc6x: 2
- User# 00023  /home/fctech/jeff/chkout/stree.org/build/srv/bin/debug/data/mark.idx: 2
- User# 00023  chnacs=y uerr=0 fileid=1219 servid=39000011x timeid=5d9b8dc6x: 463
- User# 00023  mark.idx: 463
- User# 00023  /home/fctech/jeff/chkout/stree.org/build/srv/bin/debug/data/mark.idx: 463
- User# 00023  I0000001.FCS
- User# 00023  KMAT_ERR:   indx: 0 1ax  call: 0 1bx
- User# 00023  Unique File ID error [6,4]: op=2 fid=1219 sid=39000011x tid=5d9b8dc6x: 3
- User# 00023  /home/fctech/jeff/chkout/stree.org/build/srv/bin/debug/data/mark.idx: 3
- User# 00023  WARNING: unique file ID delete failed: 8770
```

On Windows - File open may fail with error 920 (**FNAC_ERR**, the file exists but could not be accessed).

Another possible symptom that can be caused by the file open optimization is that a call to delete a file might fail internally because another connection has opened the file. In this case the function call returns success even though the file could not be deleted from disk. If the delete occurs during a failed create attempt (for example if the data file create succeeds but the index file create fails), the data file is left on disk without its resource anchor set, which causes a subsequent open of the file to fail with error **401**. If the delete is during a call to delete the file (for

example, **DELRFIL()**), the file is left on disk with its update flag set, which causes a subsequent open of the file to fail with error **14**.

Workaround: Disable the file open and close optimizations by adding the options `OPTIMIZE_FILE_OPEN NO` and `OPTIMIZE_FILE_CLOSE NO` to the server configuration file (*ctsrvr.cfg*) and restart c-tree Server. A possible side effect of disabling these optimizations is slower performance for file open and close operations.

Affected Components: c-tree Server, V11.6.1 and later.

When the file open optimization is enabled, the logic has been modified to ensure that only one thread opens a file when it is not already open. The logic has been modified so that path differences do not prevent finding the entry for the file. This means that opens of a file having the same name but a different path will be serialized when the two files are not already open, but this is expected to be an uncommon occurrence.

10.4 More Batch Operations Improve Network Traversal Performance

FairCom DB Batches provide a high capacity means to operate on multiple records at once greatly reducing network traversal time. The result for applications is higher performance. FairCom DB batches can be used in all basic database operations:

- **Add**
- **Get**
- **Delete**
- **Update**

V12 enhances batch operations in all areas as described below.

Batch Filters and Range Support Added to FairCom DB API Batch API

The efficiency of using **ctdbSetBatch()** has been greatly enhanced. Previously, several calls were required to set up a batch, run it, and tear it down. Most of these calls communicated with the FairCom Database Engine incurring network latency. Now you can set up a batch, run it, and tear it down with far fewer calls on the network. Previously a batch could take 7 calls to the FairCom Database Engine and now it can be done in as few as 2 calls.

Note: To take advantage of this feature in client/server applications, both client and server need to have the feature enabled. If one of the two does not have it enabled, an attempt to use the BAT_EXTENSIONS will fail with **NSUP_ERR**.

Note: This feature is a Behavior Change.

Batch-aware versions have been added:

ctdbSetBatchFilter()

(**https://docs.faircom.com/doc/ctreedb/ctdbsetbatchfilter.htm**)

ctdbSetBatchRangeOn()

(**https://docs.faircom.com/doc/ctreedb/ctdbsetbatchrangeon.htm**)

ctdbSetBatchRangeOff()

(**https://docs.faircom.com/doc/ctreedb/ctdbsetbatchrangeoff.htm**)

Performing a Batch Lookup (Pseudo Code)

Performing a batch lookup in the past might have looked like this:

ctdbRecordRangeOn()	←	Network hit
ctdbFilterRecord()	←	Network hit, setting filter
ctdbSetBatch()	←	Network hit
(**ctdbNextBatch()**)		
ctdbEndBatch()	←	Network hit
ctdbRecordRangeOff()	←	Network hit
ctdbFilterRecord()	←	Network hit, clearing filter

Now it can look like this:

ctdbSetBatchRangeOn()	←	No network hit
ctdbSetBatchFilter()	←	No network hit
ctdbAddToFieldMask()	←	No network hit, limits fields returned allowing more records in the same buffer size
ctdbSetBatch()	←	Network hit
(**ctdbNextBatch()**)		
ctdbEndBatch()	←	Network hit

The reduction of several network calls has proven in our testing to greatly improve the throughput capabilities with c-tree batch support.

Batch Column Filtering for Advanced Record Retrieval

Using the recently introduced *FairCom DB API field mask support*, it is now possible to filter columns in batch operations.

In ISAM, set the *fldmask* member of the PKEYREQ2 structure to be used as a "request" for the BATSETX call, using the BAT_EXTENSIONS mode. The *fldmask* must point to an array of ULONG sorted in ascending order containing the field number (DODA position) composing the partial record. It is also necessary to set the *nfields* member to indicate how many entries belong to the *fldmask* array.

Batch Mode BAT_LOK_ONE Behavior Change

BAT_LOK_ONE (a locking mode available with **DoBatchXtd()**) changes the locking strategy for a batch from holding locks for the result set until the batch is completed to only holding a record lock while the record is read.

Before this modification, BAT_LOK_ONE was ignored unless BAT_PHYS was in effect.

In V12 and later, BAT_LOK_ONE is active when BAT_PHYS and/or BAT_RET_REC or BAT_RET_FIX are part of the batch request mode.

Other retrieval strategies such as BAT_RET_POS cannot be used with BAT_LOK_ONE, and the batch request will fail with a **BTRQ_ERR** (430).

BAT_EXTENSIONS New BATSETX (DoBatchXtd) Mode

These new features have been implemented in the FairCom DB ISAM API:

1. Column filtering (i.e. return partial record based on specific column). Review *FairCom DB API - Field mask support* for more details.
2. Record filtering integrated in the BATSETX calls (record filtering was already possible but required separate C/S round trips). More details on Data Filters can be found in the *FairCom DB Developer's Guide* in *Data Filters*.
3. Index ranges integrated in the BATSETX (index ranges were already supported but required separated C/S round trips). More details on index ranges can be found in the *FairCom DB Developer's Guide* in *Index Ranges https://docs.faircom.com/doc/ctreeplus/30376.htm*.

To implement the above functionality in the ISAM layer, the BAT_EXTENSIONS mode has been used. Prior to this modification, that mode returned NSUP_ERR. This mode can be now used in the "mode" parameter of the BATSETX call.

Note: BAT_EXTENSIONS cannot be OR'd with other modes, which results in an error.

When "mode" is set to BAT_EXTENSIONS, the "request parameter" is interpreted as a PKEYREQ2 structure (or a series of LONG depending on the first two LONGS which are common to the PKEYREQ2).

```
typedef struct pkeyreq2 {
 ULONG batchmode;  /* batch mode                                          */
 ULONG batchmode2; /* batch mode2 (future use)                            */
 LONG btotal;      /* total in set                                        */
 LONG bavail;      /* number available                                    */
 LONG breturn;     /* number returned                                     */
 COUNT siglen;     /* significant key len (target/utarget size in bytes)   */
 COUNT nsegs;      /* number of segments in range                          */
 COUNT nfields;    /* number of entries in fieldmask                      */
  pTEXT target;     /* partial key target/lower-range in standalone it must */
                    /* point to a buffer as large as the key (ctnum->length)*/
  pTEXT utarget;    /* upper-range                                          */
 pNINT operators;  /* operators array for range                           */
 pULONG fldmask;   /* array of field number (DODA position)                */
 pTEXT filter;     /* filter string in case of filter                      */
} PKEYREQ2;
```

Changes compared to the PKEYREQ used when BAT_EXTENSIONS is not in use:

- *batchmode;* The Batch mode that used to be the "mode" parameter of BATSETX.
- *batchmode2;* Reserved for future use.
- *siglen;* The significant portion of the target keys in bytes.
- *nsegs;* For Range Use, this is the number of segments specified in the target keys and, as a consequence, the number of operators. Set to 0 if range is not established by the call.
- *nfields;* For Field Mask support, this is the number of fields specified in the field mask array.
- *target;* For Range Use, this is the partial key target/lower-range. In standalone it must point to a buffer at minimum as large as the key in use.
- *utarget;* For Range Use, this is the upper range. In standalone it must point to a buffer at minimum as large as the key in use.

- *operators;* For Range Use, this is the Operators array for range; set to NULL if range is not established with this call.
- *fldmask;* For Field Mask support, this is the array that specifies the field numbers (DODA position) of the field composing the partial record returned. Set to NULL if not in use.
- *filter;* For Filter support, this is the data filter to establish. Set to NULL if no filter should be established, in which case any active filter may apply.

The use of BAT_EXTENSIONS is not compatible with the following modes:

- BAT_DEL
- BAT_UPD
- BAT_INS
- BAT_RPOS
- BAT_KEYS

Improve batch read resilience to errors caused by record updates

In certain situations, a batch read could fail with errors such **160**, **153**, or **149** if it is performed at the same time another connection is updating records in the data file. This can happen if another operation locks and updates the record after the batch finds a key value that satisfies the batch criteria and before the batch locks and reads the record. Using the lock options such as BAT_LOK_RED or BAT_LOK_WRT reduce the likelihood of the error, but it can still occur.

The logic has been modified to handle these situations better. In the following cases, where the batch read fails to read a record, it now skips the record and continues the batch processing with the next key in the sequence:

1. If record read fails with error **ITIM_ERR**.
2. If the batch read is not locking records and record read fails with error **VBSZ_ERR** or **VRLN_ERR**.

In addition, the batch read operation now skips a fixed-length record that has been deleted.

10.5 Logging, Error Reporting, and Messaging

Enabling function timing no longer reduces server throughput

Prior to c-tree V11.8, enabling the collection of c-tree Server's function timings in the *SNAPSHOT.FCS* reduced the overall throughput of the server. Therefore, it was FairCom's best practice recommendation to not enable function timings in production settings, unless absolutely necessary.

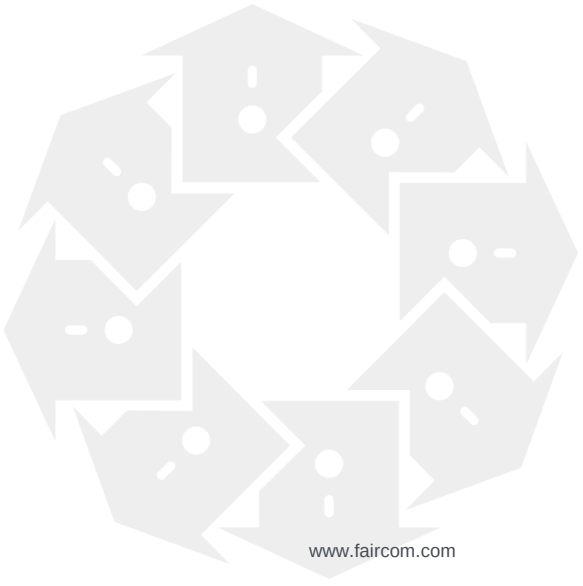
The logic has been modified to improve performance and yield more accurate results. As a result, this recommendation is no longer necessary and it's now acceptable to enable function timings for production use. We might still recommend only doing this for reasonable periods of time, for example to get baselines, or when troubleshooting an application slow down.

With this bug fix, the following values in *SNAPSHOT.FCS* are now correct; previously, they were always zero:

```
system-wide                c-tree calls: 15360578
              time waiting for requests: 1112.107
                time executing requests: 226.281
                 time to send responses: 332.027
elapsed time function-timings turned on: 81.862

system-wide   time waiting per request: 0.000072
            execution time per request: 0.000014
             time to send per response: 0.000021
```

Affected Components: c-tree Server



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.

1/30/2025

11. Index

W