test link

Update Guide

FairCom DB V12 Updates

Audience

Developers

Subject

V12 Updates and Changes in the FairCom DB - the latest release of the c-treeACE Advanced Database Engine

FairCom®

Contents

1. Welcome to FairCom DB V12

Achieve hundreds of thousands of transactions per second on a single database server and scale to hundreds of servers

FairCom DB is ideal for large-scale, mission-critical, core-business applications that require performance, flexibility and capabilities that cannot be achieved by other databases.

Predictable high-velocity transactions are a hallmark of FairCom DB. It empowers developers with easy NoSQL APIs in their favorite programming languages to process structured binary data at machine speed using custom algorithms, and it provides ANSI SQL over the same binary data for easy queries, analytics, and integration with SQL tools. Further, it provides high-speed data replication to create massively scaled solutions with a Continuum of Control from ACID compliant to eventually consistent.

FairCom DB is a unified multimodel database that gives developers a Continuum of Control to achieve unprecedented performance and that is self-tuning for the lowest total cost of ownership (TCO).

You don’t conform to FairCom DB — FairCom DB conforms to you. Don’t be held back by limitations in other databases. Meet your business needs without compromise, with the lowest TCO.

1.1 FairCom DB V12 Highlights

c-treeACE becomes FairCom DB

FairCom DB reflects an easier faster approach to database development. A cleaner product footprint (page 126). Extensively revised examples. Ready to run configuration.

A quick preview of what you'll find inside of our latest V12 release.

Go Bigger (page 3) - *Capacity limits greatly increased. More files. More space. Bigger transactions.*

Go Faster (page 8) - *File Open close performance greatly improved. Faster Indexing. More batch options. Less contention and greater concurrency.*

Develop Easier (page 79) - *100+ new and enhanced functions. New Node.js support. Server side Plugins.*

SQL (page 35) - *Dynamic SQLize of ISAM tables. Cross-database queries. Multiple value sets. Diagnostic query logging.*

Administration (page 48) - *Browser-based tools. Automatic timestamps, aggregations and purging,*

High Availability (page 62) and Data Replication (page 65) - *New synchronous replication. Client failover options. Replication Manager.*

Security (page 69) - *New master encryption key management options. Password strength enforcement.*

Latest APIs (page 79) and Functions (page 108) - *JSON, REST, MQTT.*

Compatibility changes (page 126) - *New product folder layout. New IFIL path handling. V10.1 and prior FAIRCOMS.FCS files deprecated.*

2. Go Bigger

The theme of FairCom DB V12 is "Go BIG!" We have increased FairCom DB capacity in many ways: number of open files, field size, number of fields, and number of index columns. This section explains how these features benefit your enterprise-scale and hosting applications.

2.1 Millions of Records per Transaction

Large transactions are sometimes unavoidable. Recently presented situations include a case where a large-scale database purge included many cross-referenced tables. Another case involved a full data table version update. In both cases, the transaction size challenged existing limits of performance. Ultimately, regardless of size, database changes must persist to the write-ahead log for atomicity and recovery. FairCom database optimizations make these as fast as possible rivaling, or beating other database comparisons.

FairCom DB can efficiently support large transactions without consuming excessive memory. Enhanced configuration options limit the amount of data stored in memory during a transaction. After the transaction has reached this limit, the server creates a swap file and stores subsequent data in the swap file.

Note that internal structures remain allocated, so memory use still increases somewhat. However, as record images and key values are temporarily written to the swap file, memory usage is greatly reduced when updating many records or very large records.

This feature is controlled with the following configuration options in ctsrvr.cfg:

- `MAX_PREIMAGE_DATA <limit>` sets the maximum size of in-memory data that is allocated by a transaction to `<limit>`. After this data limit has been reached, subsequent allocations are stored in a preimage swap file on disk. The default value is 1 GB.
- `MAX_PREIMAGE_SWAP <limit>` sets the maximum size of the preimage swap file on disk. If the file reaches its maximum size and a transaction attempts to allocate more space in the file, the operation fails with error **TSHD_ERR** (72). The default value is zero (meaning no limit).

Security Note: The preimage swap file contains data record images and key values that the transaction updated. Note that even if the corresponding data file or index file has encryption enabled, the *preimage swap file contents are only encrypted if the* `LOG_ENCRYPT` *configuration option is used*. If advanced encryption is enabled, the preimage swap file is encrypted using the AES-32 cipher. If not, the preimage swap file uses FairCom's proprietary ctCAMO algorithm, which is a simple masking of the contents, it is not a form of industry-standard encryption.

Changes to Hashing and PREIMAGE_HASH_MAX

When updating many records in one transaction, the update rate slowed over time. This happened even if a table lock was acquired on the table and preimage memory use was reduced with the `MAX_PREIMAGE_DATA` configuration option.

A hash table is used to efficiently search preimage space entries containing updated record images. This modification improves the hash function in these areas:

1. It no longer imposes a limit of 1048583 hash bins.
2. Improved the hash function to provide a more even distribution of the values over the hash bins.

For maximum performance, the hash function can now 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 changed from 131072 to 1048576 so that our dynamic hash of the preimage space entries can be more effective for large transactions without requiring the server administrator to remember to increase this setting.

2.2 Millions of Open Files

In licensed enterprise editions of V12 and later, c-tree supports opening more than 32,767 files. This enhancement has been designed so no application changes should be required. Each individual database connection is still limited to a maximum of 32,767 files (preserving the original definition of data and index file numbers as two-byte signed values, using the COUNT data type). Internally, new data types were introduced that support up to 2 billion files. Note that we have a compile-time limit of 1,000,000 files to reduce the memory footprint of the c-tree database engine. If you need more than 1,000,000 open files, please contact FairCom.

If opening many files, consider the possibility of using multiple c-tree database engines to host the files, rather than having one c-tree database engine hosting all the files. This is recommended because the files will compete for resources such as data and index cache, file system cache, operating system kernel memory, and transaction logs. Note the c-tree database engine is very resource friendly and it's possible to execute multiple occurrences of the engine on the same host computer (be sure to comply with the c-tree license which requires a license per installed instance).

Compatibility Notes:

Transaction log compatibility: The checkpoint log entry now contains a 4-byte number of open files rather than a 2-byte value. This means that the transaction logs created by a server without 4-byte file number support are incompatible with a server that uses 4-byte file number support, and vice-versa. When this incompatibility is detected, the database engine fails to start up with error **LFRM_ERR** (666), "incompatible log format."

The following message in *CTSTATUS.FCS* indicates a server with 4-byte file numbers found transaction logs that use 2-byte file numbers:

```
- User# 00001  Incompatible log file format [10: 45800400x 47a00490x 02200090x]
- User# 00001  L0000001.FCS
```

The following message in *CTSTATUS.FCS* indicates a server with 2-byte file numbers found transaction logs that use 4-byte file numbers (or some other new feature that this server doesn't support):

```
- User# 00001  Incompatible log file format [5: 44800400x 07a00490x 43200090x]
- User# 00001  L0000001.FCS
```

Be sure to review the server upgrade best practices in the FairCom knowledgebase. The following document lists the only recommended procedures for safely upgrading to a server that has a transaction log file format change: *Steps to Upgrade a FairCom DB Server* (https://docs.faircom.com/doc/*knowledgebase/product_upgradesteps.htm*).

Standalone Mode Utilities: Remember that this transaction log incompatibility also affects standalone mode utilities such as **ctrdmp** and **ctldmp**. These utilities must be compiled with 4-byte file number support in order to read transaction logs that were created by a database engine that supports 4-byte file numbers.

New API function: ctGetOpenFilesXtd()

The function **ctGetOpenFiles()** uses the ctFILINF structure, which contains a two-byte field for the system file number. To support retrieving the full four-byte system file number, we introduced the function **ctGetOpenFilesXtd()** (https://docs.faircom.com/doc**/ctreeplus/85127.htm**), which uses the new ctFILINFX structure, containing a 4-byte system file number field. An application that calls **ctGetOpenFiles()** uses the ctFILINF structure, so the output for each file contains a 2-byte system file number.
When the file number is greater than 32,767, this function returns **-1** for the system file number. To avoid this limitation, call **ctGetOpenFilesXtd()** instead. Note that the **ctadmn** utility has been updated to call **ctGetOpenFilesXtd()**. If that call returns an unsupported function number, **ctadmn** calls **ctGetOpenFiles()** instead.

State Variable: The c-tree connection-level state variable *isam_fil* is still a 16-bit value. A new state variable, *isam_fil32*, holds the full 32-bit file number in case of an error.

License

Support for handling more than 32,767 files is a licensed feature. The maximum number of files that can be specified by the `FILES` keyword is limited to 32,767 files when the feature is not enabled.

2.3 128 TB SQL Temp Tables

SQL queries usually require temporary storage space for sorting. Very large SQL queries require very large temporary sorting space. FairCom DB now provides up to 128 TB of temporary table space from your largest of SQL queries.

For example, the following sets the limit to 1 TB:

```
SUBSYSTEM SQL LATTE {
 MAX_STORE 1000 GB
}
```

See SUBSYSTEM SQL LATTE configuration for changing store size limit (https://docs.faircom.com/doc/sqlops/subsystem-sql-latte-config.htm)

2.4 4GB Transaction Log Space

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 potential 2GB variable-length record in a transaction. Reducing the number of logs results in less log file churning with high velocity applications leading to faster consistent transaction throughput.

2.5 64K SQL CHAR Fields

Easier handling of large SQL fields has been a common request. XML and other large text objects are frequently larger than 8192 bytes requiring LONG field handling. LONG fields can be up to 2GB and are handled quite differently than standard CHAR or VARCHAR fields as a handle to data is required. Data is then streamed from the field through the handle with calls such as **GetBytes()**. Further, LONG fields were not supported in stored procedure interfaces requiring extensive workarounds to handle large character fields.

The maximum size of a field in SQL has been increased from 8,192 to 65,500 characters. This extends the useful size for character-based data and is much easier to handle directly in SQL.

BINARY, VARBINARY, CHAR, and VARCHAR types have been expanded to a maximum size of 65,500 bytes, which means it is not necessary to use LVARBINARY or LVARCHAR for lengths between 8182 and 65500.

In addition, stored procedures have been extended (page 45) such that LONG fields can be accessed within them. However, it remains not possible to pass a LONG field into a stored procedure argument.

At the c-treeDB level, the "scale" and "precision" properties have been adjusted to be consistent with the information in the system tables.

2.6 2500 Columns per Table

The limit of SQL columns in a table is now 2,500 for FairCom DB. It was already at this limit for FairCom RTG COBOL Edition.

3. Go Faster

FairCom DB customers consistently assert the sheer speed a FairCom database provides their applications. And, it's never enough. Successful application growth continually challenges database scalability. Overall performance gains of FairCom DB and FairCom RTG are again included in this V12 release. Multiple areas from file open and close operations to index key reads have been profiled for capacity limitations. FairCom DB V12 removes many bottlenecks for additional transaction throughput.

Many of these new performance features and capacity expansions will be experienced immediately after upgrading your c-tree client and server engine. Others involve application changes and require modifications by a programmer or DBA.

Let us engage with your team and squeeze as much performance and capacity as possible from your FairCom DB application usage.

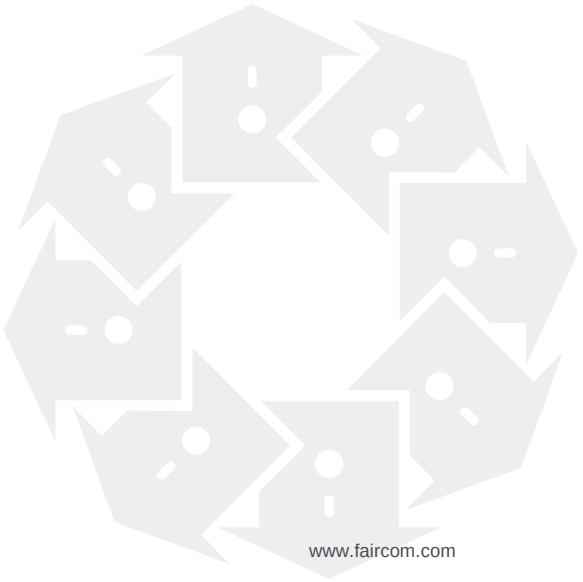
3.1 Up to 3x Faster Overall Performance

Performance has always been the hallmark of FairCom DB. Our customers consistently assert the sheer speed a FairCom database provides their applications. And it's never enough. Successful application growth continually challenges database scalability.

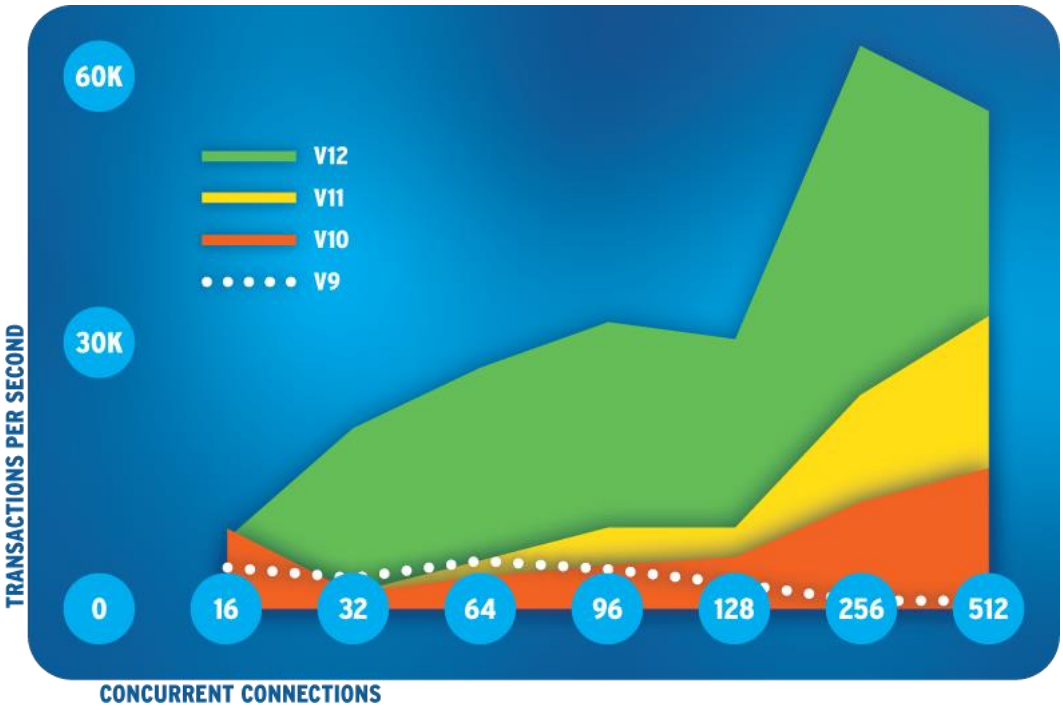
Overall performance gains of FairCom DB and FairCom RTG are again included in this release. Multiple areas from file open and close operations to index key reads have been profiled for capacity limitations. This release removes many bottlenecks for additional transaction throughput.

Many of these new performance features and capacity expansions will be experienced immediately after upgrading your c-tree client and server engine. Others involve application changes and require modifications by a programmer or DBA.

Let us engage with your team and squeeze as much performance and capacity as possible from your FairCom DB application usage.



The following graph shows the performance gains made with c-tree over the past 4 major versions using our internal test harness, **cttctx**, which uses full transaction processing (*ctTRNLOG* file mode) on all data files with the default FairCom DB Server Configuration File (*ctsrvr.cfg*) settings for each release. **cttctx** simulates a real-world application by performing record add, read, delete sequences on 23 data files each with multiple indexes. All tests were executed running Windows Server 2019.



3.2 Up to 4X Faster Indexes with Smaller Indexes Using Variable-Length Compressed Key Storage

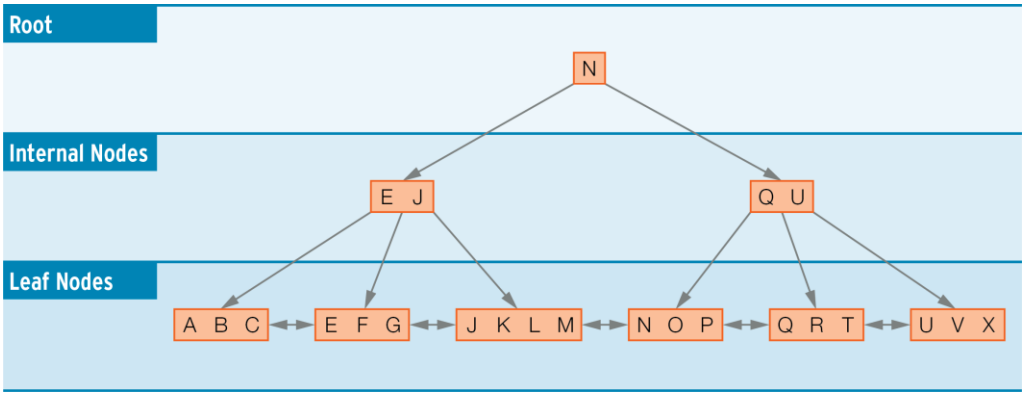
FairCom DB indexing has always been optimized for the fastest possible data access. Indexes are composed of keys. However, not all keys are composed the same. For example, consider indexing based on file paths. Path strings are highly variable in length. Indexing data based on this type of key definition requires defining a key length of the absolute possible maximum while most keys are substantially less. This introduces much wasted space into an index structure as b-tree indexes are constructed of fixed-sized nodes for efficiency of reading and writing. Each node read or write is generally performed as a single I/O operation, thus the more keys per node, the better the I/O throughput of reading keys in an index. Further, large variably sized keys force a large index node size to enforce a three key per node minimum. FairCom has addressed this wasted space challenge for greatly reduced index sizes, leading to less I/O and ultimately, increased performance.

In FairCom DB, we obtained considerable index performance for large key lengths. A new Variable-Length Key Compression ("Vlen Keys") demonstrated up to 6x reduction in storage space and up to 28x faster performance in testing with the FairCom DB Server. The ideal

scenario for this level of performance gain involves a reasonable length field (perhaps 32 bytes or larger) that will be roughly ½ of the maximum field length or less for most records.

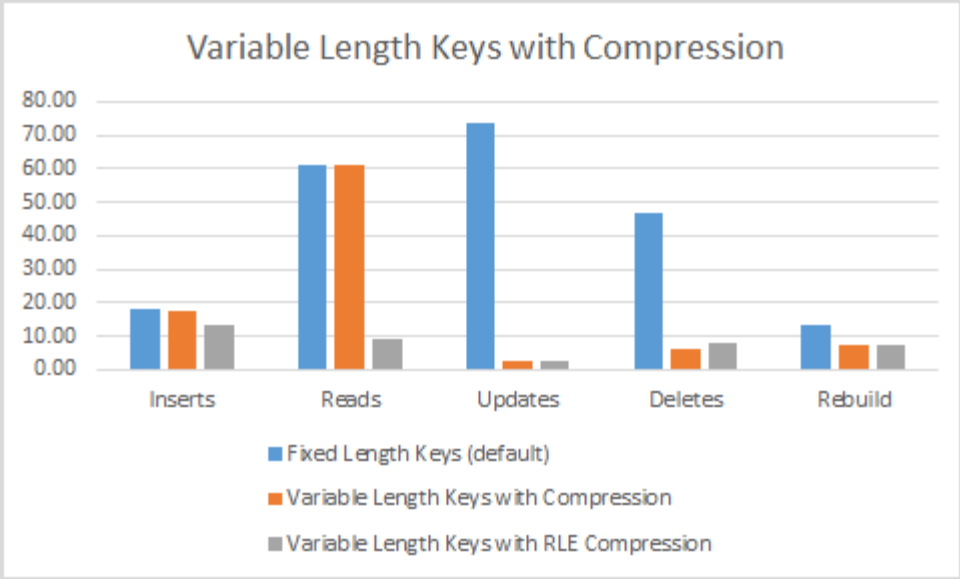
By default, FairCom DB indexes store keys as fixed-length values. This requires few CPU cycles however, increases storage space and resulting I/O. Taking advantage of the fact CPU processing is orders of magnitude faster than I/O, we created a new compressed, variable-length index structure significantly reducing storage space along with a concomitant reduction of I/O when reading and updating index data.

This effort required advanced optimizations to our Index Node/Page handling. Now we have the ability to store more index entries per Node/Page, resulting in more "bang for the buck" on each disk I/O. These internal improvements do not affect your existing application code. To take advantage of this new advanced index compression, simply rebuild indexes with the new settings detailed in the sections that follow.



Variable-Length Key Performance Results

Substantial improvements in performance were seen when using key compression. The following tests were conducted using the FairCom DB Server on a table with 1 million records and a single index over a 1,000-character field that contained a file name with long paths.



- Insert column is the time in seconds to insert 1,000,000 records.
- Read column is the time in seconds to read 100,000 random records.
- Update column is the time in seconds to update 100,000 random records.
- Delete column is the time in seconds to delete 100,000 records (no duplicates).
- Rebuild column is the time in seconds to rebuild the full 1,000,000 record table.

Option to Automatically Enable c-tree Key Compression When Creating an Index

FairCom DB now supports a configuration option that causes all indexes that contain a key segment that uses the Unicode key segment mode to be created with c-tree leading and padding key compression. By default, this option is off. To enable this option, add `UNCSEG_KEYCOMPRESS YES` to *ctsrvr.cfg*. This option can also be enabled or disabled at runtime by calling the **ctSETCFG()** option or using the **ctadmn** utility's option to change a server configuration option value.

c-treeDB API Compression

New index node types provide options for reducing index size. [01/2024]The new NAV (c-treeDB) index modes are as follows:

- **Variable-Length Keys with Compression**: `CTINDEX_VARIABLE` index mode provides variable length keys with padding compression to squeeze more capacity into each page block. Recommended for indexes over variable length data.
- **RLE Compression**: CTINDEX_VARIABLE_RLE index mode uses variable length keys with a simple RLE key compression of bytes 0x0, 0x20, 0x30 resulting in shorter index entries and smaller index files. Recommended for indexes over variable length data with multiple segments, binary data, or large numeric data types (8 byte integers).

Example

The following example demonstrates how to use one of these modes:

```
pIndex = ctdbAddIndex(hTableCustMast, "cm_custnumb_idx", CTINDEX_VARIABLE_RLE, NO, NO);
```

For more information, see **ctdbAddIndex** (https://docs.faircom.com/doc**/ctreedb/ctdbaddindex.htm**) in the c-treeDB documentation.

ISAM API Compression

[01/2024]New "Alternative Key Types" have been added to support additional compression options. These key types support the compression of indexes with the ISAM API:

- `KTYP_VLENGTH (0x608)` - Eliminates the key padding bytes on disk without the high performance costs of the legacy `COL_SUFFIX` key compression. This is suggested for indexes over variable length fields that may have a lot of variation in the length such as street address, and filesystem paths. Anyone using `COL_SUFFIX (8)` should investigate this improved alternative.
- `KTYP_VLENGTH_SRLE` (0xE00) - Simple RLE compression without the high performance costs of legacy key compression. This is suggested for indexes over fields that may have a lot of repeating binary 0, ASCII space, or ASCII 0 values. This may be beneficial for keys over many data types such as 8 byte integers, binary data, or other variable length data.

These `KTYP_` values are bits that can be set in the *IIDX.ikeytyp* for each index.

See the section in the *FairCom ISAM for C Developer's Guide* titled *IIDX Structure* (https://docs.faircom.com/doc/*ctreeplus/30859.htm#o30861*).

Utilities to Confirm Index Compression Modes

The file information utility, **ctinfo**, displays index compression modes as included in the key type when displaying index information.

Example

```
IIDX #2 {

        /* key length              */ 4,
        /* key type                */ 2048, (0x0800 = KTYP_KEYCOMPSRLE)
        /* duplicate flag          */ 0,
        /* null key flag           */ 0,
        /* empty character         */ 0,
        /* number of segments      */ 1,
        /* r-tree symbolic index   */ cm_custnumb_idx,
        /* alternate index name    */ 0000000000000000,
        /* alternate collating seq */ 0000000000000000,
        /* alternate pad byte      */ 0000000000000000,

};
```

3.3 Faster Indexing from Locking, Node Pruning, and Sorting Optimizations

Improved Performance of Server Index Node Lock

The FairCom Server logic has been enhanced to improve performance for concurrent connections that are updating the same index file. The performance of FairCom Server index node lock request collision handling has been improved by adjusting the logic to be more efficient.

This change has also been applied to data record write and read lock collision handling.

Faster Index Additions from ctdbAlterTable() (https://docs.faircom.com/doc/ctreedb/ctdbaltertable.htm)

When **ctdbAlterTable()** is called to add new indexes without specifying a physical index name, c-treeDB adds them as members to the last physical index. To accomplish this, it dropped all existing indexes and rebuilt them including the new members.

ctdbAlterTable() now avoids rebuilding existing indexes whenever possible and simply adds new index members. This makes **ctdbAlterTable()** much faster when adding new indexes in many cases.

This change also improves IFIL content by avoiding storing unnecessary index names.

Improved Scalability for High Concurrency Index Leaf Node Reads

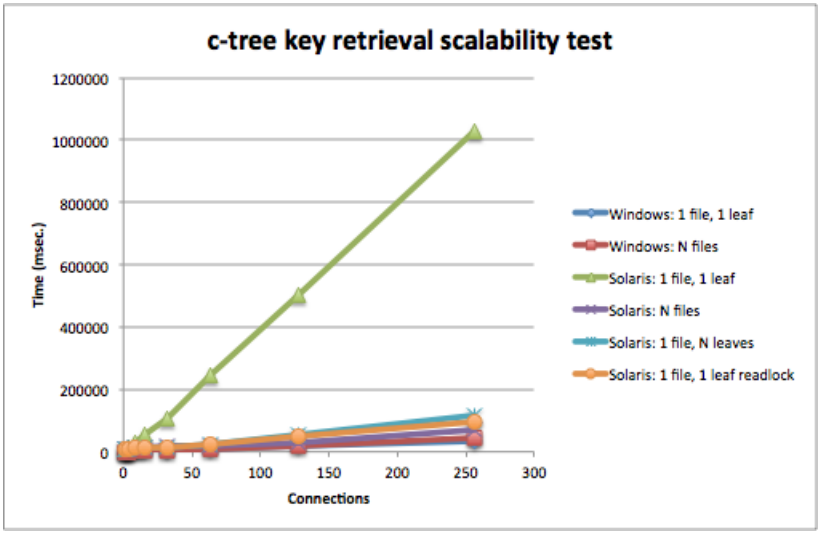
Performance analysis identified contention on index leaf node reads with many concurrent connections. A new configuration option improves scalability under these conditions. Specific testing on a Solaris system with many CPUs clearly shows large improvement. This modification affects the following index types:

1. An index that uses key compression.
2. An index that is under transaction control.

With this configuration enabled, FairCom Server acquires a read lock on a leaf node if the index does not use key compression. If it finds that the node contains transaction marks, it releases the read lock and acquires a write lock on the node.

The FairCom Server configuration option `LEAF_NODE_READ_LOCK` accepts values of `YES` (to turn on this feature, which is the *default*) and `NO` (to turn off this feature). This configuration option is dynamically configurable at runtime by calling the **ctSETCFG()** function or using the **ctadmn** utility's option (menu option 10, and then menu option 10, Change the specified configuration option, again) to change a server configuration option value.

The graph below shows specific performance improvement. The green line is our starting point with a proprietary test, which is roughly 1,000 seconds at 250 concurrent connections. After our change, the performance characteristics on Solaris is around 100 seconds, an order of magnitude improvement.



Background Load on Index Creation

Background key loading can improve performance when adding new indexes. A new background loading mode can be OR-d in with an existing key mode from a **ctdbAddIndex (https://docs.faircom.com/doc/ctreedb/ctdbaddindex.htm)()** call indicating initial index loading must be done in the
background.

Once **ctdbAddIndex()** is called, **ctdbAlterTable()** initiates the changes to the file and begins the load once the new index is successfully created.

Check the status of index loading with the new **ctdbCheckIndexBackgroundLoad()** (https://docs.faircom.com/doc**/ctreedb/ctdbCheckIndexBackgroundLoad.htm**) API, useful for monitoring index loading on tables.

Index Node Prune Feature - Ability to Deactivate Node Pruning

FairCom DB's delete-node thread prunes empty nodes from index files in the background. It densely packs key data for optimal index performance. To prevent index corruption, it runs during idle times and opens the index file exclusively, which prevents external processes from opening it.

To disable the "index node prune" feature, for example to allow an external process to open an index file, use the following keyword:

```
COMPATIBILITY NO_DELNOD_QUEUE
```

The above keyword prevents the delete-node thread from running, which prevents automatic index optimization, however, it allows external processes to open the index file.

Another keyword, `COMPATIBILITY KEEP_EMPTY_NODE_ON_READ`, prevents empty nodes read from disk being added to the delete queue. This is the V2/V11 and earlier behavior.

Extend Leaf Node Read Lock Optimization to Moving Right or Left at Leaf Level

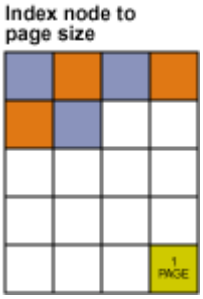
When key lookup operations move right or left at the leaf level, the call to search the index cache for an index buffer that holds the specified node was not taking advantage of the leaf node read lock optimization. The logic has been modified to do this. This change could boost performance of key lookups when an index has empty leaf nodes for example.

3.4 Up to 15% Faster with Increased Default Index Page Size

Background

Index node page size is an important performance attribute to consider. Page size is equal to a node size within the index binary tree. Each index node is a single I/O to and from persisted storage into a single allocated buffer cache page. There is a direct one to one mapping of cache pages to index nodes.

Figure 1: Data Length to Page Size



Ideally, the page size should be configured as close to the I/O read/write block size used by the operating system. In many OS environments this is 4K. Selected storage systems allow tuning this system parameter for optimum database performance. However, a larger c-tree page size can still take advantage of multiples of system block I/O. Modern storage systems generally have enough redundant capacity that I/O operations are completed even when power is lost. It is still recommended that uninterruptible power supplies (UPS; battery backup) are used to ensure absolute data integrity at all times.

The default page size has been 8192 bytes since V8.14 for FairCom servers. The default for V12 has now been increased to 32768.

Larger Page Size Benefits

Numerous performance tests have shown increasing the default index page size (standalone *sect* parameter) can bring big gains to overall performance. Increasing the page size brings several beneficial properties to an index:

- Larger nodes contain more keys per node which can be read per I/O reducing overall I/O.
- Larger nodes allow more keys per tree level reducing overall tree depth. Shallower tree depth results in faster index searching.
- Larger nodes allow more efficient storage of larger keys. (c-tree enforces a three key per node minimum.)

32K Page Size Default

For V12, FairCom has increased the default server page size to 32768 bytes. Internal FairCom testing consistently shows up to 15% gains in overall throughput with a 32K page size. This confirms much evidence from the field with high performance applications using larger page sizes.

Page size is set with the `PAGE_SIZE` (https://docs.faircom.com/doc/`ctserver/page-size-config.htm`) parameter in *ctsrvr.cfg*.

```
PAGE_SIZE  32768
```

Be sure to review your current c-tree page size usage compared with this new default. FairCom strongly encourages thorough performance testing using the new page size with your specific application based on the compatibility information below.

Warning: Changing the `PAGE_SIZE` (https://docs.faircom.com/doc/`ctserver/page-size-config.htm`) is a
maintenance task that should be done carefully and with a full reliable backup. Practice on a copy of your data and server folder until you are confident with the results. For procedures, see *Adjusting PAGE_SIZE* (https://docs.faircom.com/doc/*FairCom-Installation/AdjustingPAGE_SIZE.htm*) in the *FairCom Installation Guide*.

Note: A file created with a larger `PAGE_SIZE` cannot be opened by a server with a smaller `PAGE_SIZE`.

3.5 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 c-treeDB 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()

Performing a Batch Lookup (Pseudo Code)

(https://docs.faircom.com/doc**/ctreedb/ctdbsetbatchrangeoff.htm**)
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 *c-treeDB field mask support* (page 21), 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 *c-treeDB - Field mask support* (page 21) 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 /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

3.6 Field Mask Support Added to c-treeDB

In V11.6.1 and later, c-treeDB introduces a new concept used to optimize client/server communication by retrieving only significant record portions - fields - instead of the entire record.

This feature requires a way to specify which of the record's fields are significant and will be accessed. When retrieving a record, c-treeDB attempts, when possible, to retrieve only the portion that will be accessed. This reduces the number of bytes transmitted. In the case of batches, it increases the number of records retrieved in a single round-trip with a single batch call.

To achieve this objective, we implemented the concept of a "field mask." A field mask defines which are the significant fields. It is activated when the first field is added to the mask. Once active **ctdbWriteRecord** and **ctdbDeleteRecord** fail. The field mask cannot be changed when a batch is active. Only the fields specified in the field mask can be accessed using the **ctdbGetFieldAs*** functions.

New functions:

- A field mask is activated and populated by calling **ctdbAddToFieldMask() (https://docs.faircom.com/doc/ctreedb/ctdbaddtofieldmask.htm).**
- A field mask is deactivated and emptied by calling **ctdbRemoveFieldMask()** (https://docs.faircom.com/doc**/ctreedb/ctdbremovefieldmask.htm**).
- Check if a field mask is active with **ctdbIsFieldMaskOn()** (https://docs.faircom.com/doc**/ctreedb/ctdbisfieldmaskon.htm**).

3.7 Parallel Data Processing Scales Performance

Divide and Conquer Multiple Defined Fixed-Length File Regions

When performing a full-table scan of a large data file, a significant speed-up can be achieved by running multiple threads to process different parts (regions) of the data file simultaneously.

In V12 and later, **GetFileRegions()** (https://docs.faircom.com/doc**/ctreeplus/getfileregions.htm**) divides the data record offsets for a specified number of regions for a *fixed-length record data file*. The caller specifies the number of regions and the function populates an array of offset-length pairs. Each offset-length pair defines a *region*, which is simply a set of records in the data file. An application can process
a data file's records in parallel by assigning a different thread to process each region.

GetFileRegions() (https://docs.faircom.com/doc/ctreeplus/getfileregions.htm)

3.8 Faster File Open and Close under High Concurrency

Opening and closing files is an expensive FairCom Server I/O activity. Specific application classes stress this area more than others. For example, FairCom RTG COBOL applications frequently open and close many files. Analysis of these operations has identified specific areas for improvement, notably with high concurrent operations as these application increase connections.

Performance of FairCom Server concurrent file open and close operations has improved up to 77% in specific test cases. These improvements were especially noted when scaling on large systems, such as those with many CPU cores and concurrent database connections.

Scalability Test Results

Test programs were run on Windows 10, Solaris 11, and Linux CentOS 6, with 128 connections repeatedly opening and closing an ISAM file set consisting of one data file and three index files (a host index plus two members). Note: 2000 iterations were run on Windows and CentOS 6; only 200 iterations were run on Solaris (due to the slower CPU speed of the Solaris test box).

The number of files was varied between 1 and 128 as shown in the table below. Note that all connections opening the same file is the best case because much of the time the file is already open. Each connection opening its own file is the worst case because each open and close physically opens and closes that connection's file.

Original:

Files	Windows 10	Solaris 11	Centos6
1	2.022	2.837	3.067
2	2.231	3.654	2.902
4	2.558	14.189	2.915
8	6.439	11.581	2.698
16	18.607	22.002	3.139
32	27.337	32.489	6.589
64	60.170	50.911	11.589
128	113.519	78.803	21.287

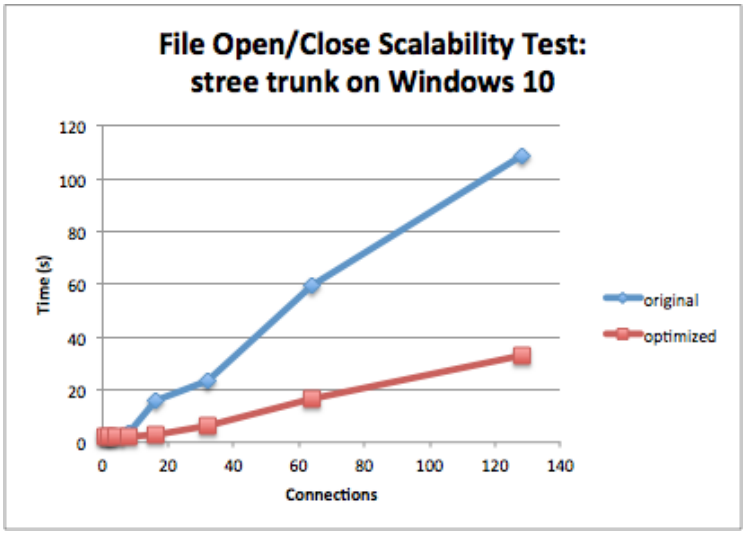
With file open/close optimizations:

Files	Windows 10	Solaris 11	Centos6
1	1.804	1.894	2.713
2	1.795	1.771	2.687
4	1.940	2.149	2.597
8	2.036	2.706	2.575
16	2.357	3.686	2.492
32	5.241	7.010	5.355
64	12.379	13.462	10.452

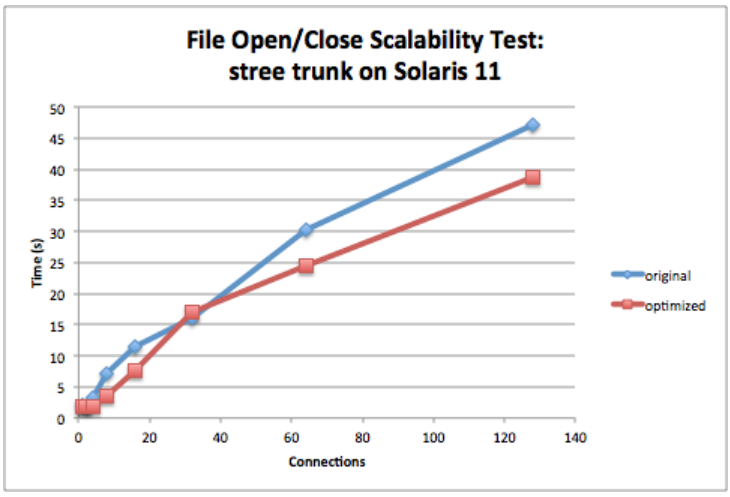
Files	Windows 10	Solaris 11	Centos6
128	25.849	27.322	18.778

For the worst case (128 connections on 128 files), the improvements are:

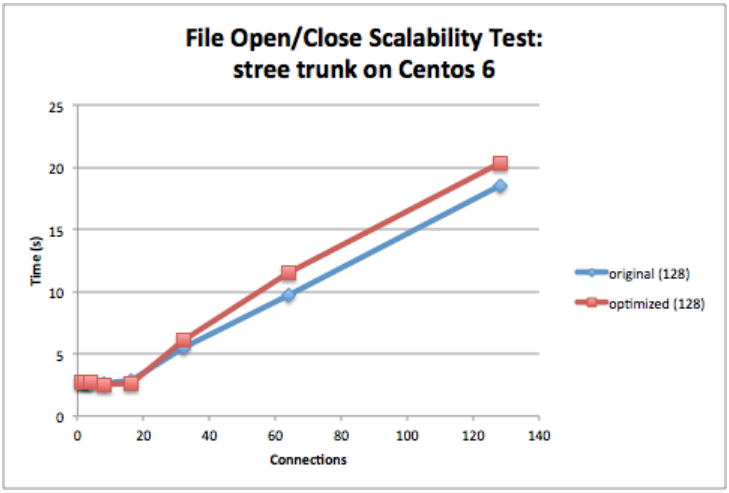
Windows 10: **77% reduction** in time (from 113.5 sec. to 25.8 sec.)



Solaris 11: **65% reduction** in time (from 78.8 sec. to 27.3 sec.)



Centos6: **12% reduction** in time (from 21.3 sec. to 18.8 sec.)



These modifications made a smaller relative difference on CentOS 6 than on other systems as the original performance on CentOS 6 (21.3 sec.) was already close to the time of the optimized performance on the other systems (25.8 sec. and 27.3 sec.).

File Open / Close Configuration Options

```
PENDING_FILE_OPEN_RETRY_LIMIT <limit>
```

The pending open retry limit is set by the configuration option `PENDING_FILE_OPEN_RETRY_LIMIT`, which defaults to 0 (wait forever). Normally, we do not expect a file open or create to exceed the pending file open retry limit, but this limit is provided to ensure the calling function returns an error after a limited number of retries in the unexpected case that a file remains in a pending open state for a long period of time.

```
OPTIMIZE_FILE_OPEN <value>
OPTIMIZE_FILE_CLOSE <value>
```

These configuration options accept values of YES (to enable the optimization) and NO (to disable the optimization). Both of these optimizations default to YES. These options can only be set in the configuration file. They cannot be changed at run time.

The following messages in *CTSTATUS.FCS* indicate that these optimizations are on:

```
File open optimization is enabled.
File close optimization is enabled.
```

The following messages in *CTSTATUS.FCS* indicate that these optimizations are off:

```
File open optimization is off.
File close optimization is off.
```

Note: This change is a server-side only change and it is not mandatory to relink your client applications to benefit. However, it is FairCom's best practice recommendation to always relink all client-side applications such that the client version always matches the FairCom Server process whenever possible.

PENDING_FILE_OPEN_RETRY_LIMIT is Configurable at Runtime

It is possible to change the `PENDING_FILE_OPEN_RETRY_LIMIT` server setting at runtime in the usual ways (**ctadmn** and the **ctSETCFG()** API function).

Error **POPN_ERR** (1130) is not typically expected during normal operation. If this error occurs, increasing the `PENDING_FILE_OPEN_RETRY_LIMIT` setting to a larger value might prevent or reduce occurrences of that error; setting it to 0 definitively prevents the error but open requests otherwise failing may wait for long time.

See Also

DIAGNOSTICS POPN_ERR (https://docs.faircom.com/doc/ctserver/diagnostics-popn-err-config.htm)

POPN Error 1130

The new c-tree error code **POPN_ERR** (1130) is returned by a file open or create function when the operation cannot be completed because the file unexpectedly remains in a pending open state for the maximum number of pending open retries.

System File ID Lists

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.

Faster File Open and Close with Large Numbers of Open Files

To speed file-open checks, FairCom Server stores names of open files in a hash table. As long as filenames are evenly distributed over the hash bins, each bin should contain relatively few entries. An updated hash function now more evenly distributes filenames for scalability.

- On Windows testing, test case performance with 128 threads went from *11970 ops/sec to 52958 ops/sec.*
- On Solaris testing, test case performance with 128 threads went from *2598 ops/sec to 3723 ops/sec.*

Faster Return for Files Not Found on Open

Profiling found unnecessary time spent in open attempts on files that don't exist. Open operations now explicitly check if the file exists, and if not, immediately returns an error. Previously, the open functionality continued processing before returning with the final error condition.

For a file that does not exist, the file open function returns error code of **FNOP_ERR** with system error code (*sysiocod*) **ERROR_FILE_NOT_FOUND** on Windows and system error code **ENOENT** on Unix systems. Applications should always check this *sysiocod* value for full **FNOP_ERR** error context.

3.9 Improved Performance Reassigning Transaction-Controlled File's ID

When opening files, FairCom Server assigns a unique file ID. These IDs change as files are opened and closed over time. Keeping track of those changes is important when reconstructing open and close events after a recovery event for transaction controlled files.

This enhancement (off by default) relieves slow file open and close operations when transaction-controlled files have their file IDs reassigned.

When a file ID of a transaction-controlled file is reassigned, c-tree scans the transaction log from the most recent checkpoint forward looking for a log entry that references the file's file name and file ID.

Scanning transaction logs can take significant time, especially when using large transaction logs. This is done while holding a global mutex, which has the effect of pausing other file open or close requests while the transaction log is being scanned. With the introduction of the system file ID feature discussed below, instead of needing to scan the log, a new log entry is now inserted signaling the file ID reassignment, eliminating the need to finish scanning the remainder of the log. This can offer a substantial time saving during the file open and close operation.

Compatibility of Transaction Logs

Enabling this new behavior breaks compatibility of transaction logs with versions of c-tree Server or standalone utilities that do not recognize this new log entry. For this reason, enabling the new behavior requires the following action depending on the operating mode:

c-tree Server:

To enable the new behavior, add the configuration option `LOG_FILE_ID_CHANGE YES` to *ctsrvr.cfg* and restart c-tree Server. To disable the new behavior, use `LOG_FILE_ID_CHANGE NO`. The default is `YES`.

Standalone Mode:

To enable the new behavior, call **ctSETCFG()** before initializing c-tree as shown below:

```
NINT retval;
retval = ctSETCFG(setcfgCONFIG_OPTION, ""LOG_FILE_ID_CHANGE YES"");
```

After c-tree has been initialized, the setting cannot be changed. A call to **ctSETCFG()** to change the setting after c-tree has been initialized returns error code **454** (not supported).

Compatibility Notes

1. Transaction Log Compatibility

When the database engine uses the new log entry (even if the log entry is not present in the transaction logs), the logs are marked as incompatible with a database engine that does not support the new log entry. If an incompatible database engine attempts to use incompatible transaction logs, database initialization fails with c-tree error 666 (**LFRM_ERR**, incompatible log format) and a message is logged to *CTSTATUS.FCS* indicating the reason for the log incompatibility.

Example log messages for this situation:

```
 - User# 00001 Incompatible log file format: log is using unrecognized options [5: ct_logsup=47a00490x
log_defrel=62200090x unknown=20000000x]
 - User# 00001 L0000001.FCS
 - User# 00001 Make sure old server shutdown cleanly, and remove old logs before re-starting server
 - User# 00001 Could not initialize server. Error: 666
 - User# 00001 O1 M0 L74 F666 P0x (recur #1) (uerr_cod=666)
```

2. Limitation on PUTHDR() call with a mode of ctTIMEIDhdr or ctUNIQIDhdr

A call to **PUTHDR()** with a mode of *ctTIMEIDhdr* or *ctUNIQIDhdr* for a transaction-controlled file that has been updated in the current transaction now fails with error **TEXS_ERR**. We do not allow changing a file's file ID during a transaction that has updated the file because of possible effects on automatic recovery.

3.10 Windows File System Compression Support

The FairCom Server configuration option `FILESYS_COMPRESS_FILE` can be used to enable file system compression on files whose names match the specified filename, which can include wildcard characters. The compression is enabled at the time that the file is created. This option only affects data and index files, not transaction logs or *CTSTATUS.FCS*.

Limitations

Currently, support for file system level compression is implemented only on Windows systems, using the built-in compression feature of the NTFS file system.

https://docs.microsoft.com/en-us/windows/win32/fileio/file-compression-and-decompression

Example

```
FILESYS_COMPRESS_FILE custmast.dat
FILESYS_COMPRESS_FILE custordr.dat
FILESYS_COMPRESS_FILE ordritem.dat
FILESYS_COMPRESS_FILE itemmast.dat
```

3.11 Faster Connections and App Communication

Shared memory performance enhancement for all Unix platforms

A shared memory performance enhancement has been enabled for all Unix platforms, starting with the base c-treeACE V11.6 line. The following changes have now been well proven in production use since late fall of 2017.

The Unix/Linux shared memory communication protocol has been changed to improve performance by improving the internal spin operation to be more efficient, especially for relatively short database operations.

Compatibility Note: It is important to recompile the client due to this shared memory change. A server that uses this modified shared memory protocol only supports shared memory connections from clients that also use this new enhanced protocol. If an older client (pre-V11.6) attempts to connect, it will fail with error **SHMC_ERR** (841) and the server will log the following message to *CTSTATUS.FCS*:

```
Fri May 26 12:13:07 2017
 - User# 00016  FSHAREMM: The client's shared memory version (3) is not
compatible with the server's shared memory version (4)
```

Performance enhanced by reducing calls to system time() function

Each function call made by a c-tree client required calling the **time()** system function to get the starting time of the current request. This revision changes that architecture to reduce the number of these calls, thereby improving performance of each FairCom DB function called by a client application.

New add record blocking lock mode avoids network traversals

When a table lock is in effect, a record add call by a connection that is not holding the table lock fails with error **DLOK_ERR** (42) and sysiocod of **DLKT_COD** (-1024) if that connection has not enabled the ISAM blocking write lock mode. Calling **LKISAM(***ctENABLE_BLK***)** before adding the record avoids the error, but it requires an additional call to the server, plus another call after the record add call to restore the original lock mode.

The c-tree client library and FairCom Server now support a new **SetOperationsState()** mode that can be used to enable the blocking write lock mode during the record add call and then restore the original lock mode, all in a single call to the server.

To use this feature, call **SETOPS(***OPS_LOCKON_ADD_BLK, OPS_STATE_ON***)** before calling the record add function (**ADDREC()** or **ADDVREC()**). The **SETOPS()** call sets a bit on the client side and returns to the caller without making a call to the server. This bit is passed to the server on the next record add call. The server and client library turn off this mode when the record add call completes, so it applies only to the first record add call that is made after turning on this option. For this reason, it's recommended to enable this mode right before the call to add the record to which you want this mode to apply.

The record read functions that check and turn off the OPS_LOCKON_GET and OPS_LOCKON_BLK modes ignore this new mode. The record add functions that check and turn off the OPS_LOCKON_ADD_BLK mode ignore the OPS_LOCKON_GET and OPS_LOCKON_BLK modes.

Note: Both the client library and the server must support this feature to use it. If the client supports the feature but the server does not, a call to **SETOPS(***OPS_LOCKON_ADD_BLK, OPS_STATE_ON***)** fails (uerr_cod is set to **NSUP_ERR**, 454) and there is no effect on the next record add call. For this reason, it is good practice to check for a failed call to **SETOPS()**. Because the return value from **SETOPS()** is the present state of the operation status word, not an error code value, the proper way to check for a failed call to **SETOPS()** is to check the value of

uerr_cod after calling **SETOPS()**. If uerr_cod is 0 (NO_ERROR), the call succeeded. Otherwise, the call failed with the error code whose value is stored in uerr_cod.

Example:

```
SETOPS(OPS_LOCKON_ADD_BLK,OPS_STATE_ON);
if (uerr_cod)) {
    printf("Error: Failed to enable auto blocking lock mode on next record add: %d\n",
        uerr_cod);
} else {
    printf("Enabled auto blocking lock mode on next record add.\n");
    ADDVREC(...);
}
```

Affected Components: c-tree client library, c-tree Server

Compatibility: Note that new client and server are both required in order to use the new feature. Old clients are compatible with new server. (They are not aware of the new feature.) New client returns a **NSUP_ERR** error if the new feature is used with an old server.

Performance enhanced by client function to get information for multiple connections in a single call

When retrieving connection information for many connected c-tree clients, FairCom DB Monitor could take a very long time to return. The bulk of this time was spent in client-by-client connection returns. To reduce this expensive network traffic, a new function was added to obtain all connection information in a single API call. This function is available for all client applications.

FairCom Server now supports a function that returns information for multiple connections in a single call. Prior to the availability of this function, a function call from the client to the Server was required for obtaining information about each connection of interest. This new approach greatly reduces network traffic.

ctGetConnectionInfo

Function prototype:

```
extern ctCONV NINT ctDECL ctGetConnectionInfo(NINT versn,pctCONINF pconninfo,pNINT pnconnections);
```

Version 1 structure:

```
/* Output data format for ctGetConnectionInfo() function: */
typedef struct ctconinf {
 COUNT utaskid; /* internal task id  */
 COUNT uactflg; /* active request indicator */
 LONG umemsum; /* user memory   */
 LONG ulogtim; /* logon time   */
 LONG ucurtim; /* current time   */
 LONG urqstim; /* time of last request  */
 LONG utrntim; /* time of last TRANBEG  */
 LONG unbrfil; /* number of open files  */
 LONG urqstfn; /* last request function # */
 ULONG sipaddr; /* client IP address  */
 ULONG sip6addr[4]; /* client IPv6 address  */
 TEXT unodnam[32]; /* node ID information  */
 TEXT uname[32]; /* user id string  */
 TEXT ucominfo[32]; /* communication info  */
 TEXT urqst[32]; /* last request function name */
 } ctCONINF, ctMEM *pctCONINF;
```

Example:

```
 NINT rc,nconnect;
 ctCONINF conninfo[32];

 nconnect = 32;
 if ((rc = ctGetConnectionInfo(ctCONINF_VERS_V01,conninfo,&nconnect))) {
  printf("Error: %d\n", rc);
 } else {
  printf("Current number of connections: %d\n", nconnect);
 }
```

Note: The caller must be a member of the ADMIN group; otherwise the function returns error **LADM_ERR** (589).

Socket wait interval can be used to check server failover status

The client library now supports setting a time interval in seconds after which a socket receive call checks the server failover status. If a failover has occurred, the c-tree function call returns error **SERVER_FAILOVER_ERR** (1159). If no failover has occurred, the socket receive call waits again for the specified time interval. To use this feature, call this function:

```
NINT retval;
retval = ctSetCommProtocolOption(ctCOMMOPT_SOCKET_WAIT_INTERVAL, socketWaitInterval);
```

where *socketWaitInterval* is a string that holds the desired time interval in seconds, for example, "5" to indicate 5 seconds.

This function can be called at any time to change the setting. A c-tree instance must have already been initialized before calling this function. It affects the current c-tree instance.

Note: The socket timeout value, which has been supported for a number of years, can now be set using this function instead of having to modify a field of the c-tree global variable structure directly. For example, to set the socket timeout to 10 seconds call:

```
retval = ctSetCommProtocolOption(ctCOMMOPT_SOCKET_TIMEOUT, "10");
```

Affected Components: c-tree client library

3.12 More Concurrency with Less Lock Contention

It was reported that FairCom Server was hitting a throughput limit due to the overhead imposed by internal lock counters (e.g., **ctstat.exe** *-filelocks*, **SnapShot()**, etc.). These counters used a strong mutex when updating lock statistics.

Atomic operations are much more efficient for these global update counters and are now used instead. In the reported case, performance gains after this change were close to doubling throughput (approximately 33K TPS to nearly 65K TPS).

Note: When this optimization is enabled, which is on by default, the FairCom Server `LOCK_MONITOR` keyword is no longer available. The lock statistics available through **ctstat** and **SnapShot()** are still available and are more complete than the `LOCK_MONITOR` keyword.

Note: This change is a server-side only change and it is not mandatory to relink your client applications to benefit. However, it is FairCom's best practice recommendation to always relink all client-side applications such that the client version always matches the FairCom Server process whenever possible.

3.13 OpenSSL Now Provides Default Faster AES Encryption

FairCom now supports an updated AES algorithm based on the OpenSSL standard by default. Previous algorithm implementation can be restored by specifying in *ctsrvr.cfg* the keyword `OPENSSL_ENCRYPTION NO`.

This change is expected to provide security and, with internal testing, we have seen an up to 20% performance improvements.

This modification changes file passwords encrypted on the client-side for secure transmission to use OpenSSL.

3.14 Experience Extreme Speed with an In-Process Database

The FairCom Database supports several operational models in addition to the traditional client/server model. The Server DLL model essentially allows you to compile the server into your application. This is the latest generation of the Bound Server model in which your application calls the server as a DLL. In the latest Server DLL moel, it is linked as a static library.

FairCom is pleased to provide you the ability to link your own functionality directly into FairCom's industry proven database server with the FairCom SDK. This unique level of control sets FairCom apart from other alternatives by offering you the true power and flexibility required by your most demanding database server applications. With additional source code availability, no other database server vendor provides this level of access and control.

The Server DLL (based on the Bound Server Model introduced in earlier releases) provides the advanced engineer a proven database engine in the form of a multi-thread safe library. FairCom's multi-threaded API and available communication logic give the serious engineer the necessary tools to create custom application-specific Servers.

This model allows you to tightly bind the FairCom Database Engine with your own application to create a powerful multi-threaded package.

Note: This model requires significant initial design efforts on the part of an experienced C programmer. Your application code implements the fundamentals of a multi-threaded FairCom Database Engine, such as threads, security, and any other requirements of your special application.

The FairCom Server DLL Development Program requires a signed distribution license prior to deployment. This model "takes the lid off" FairCom's advanced database server and gives the developer the opportunity to use FairCom's sophisticated database engine for custom needs. For additional information, contact your nearest FairCom office to discuss your requirements.

Benefits

- Linkable library allows you to link your own needs into the multi-thread safe FairCom Database Engine.
- Great technology to use for your web application servers. Bind web applications directly with the FairCom Database Engine and eliminate the overhead of inter-process communications.
- Supports all FairCom API functions and offers exceptional performance.

Considerations

- Requires engineering expertise to design and implement the FairCom Database Engine code.
- Requires a distribution license for production use.

Server DLL sample makefile in drivers folder

The folder named *ctree.srvdll* in the *drivers* folders provides examples of how to link with the Server DLL. This folder contains a makefile for each platform. *Include* folders have been added for both SQL and ISAM capabilities. The makefiles demonstrate linking to the Server DLL provided in the *bin/ace/sql* folder. The proper header files are provided to ensure consistency.

Function to pass FairCom DB configuration file contents in buffer

The c-tree Server DLL (shared library on Unix) supports a way of passing c-tree Server configuration options to the database engine rather than requiring a configuration file. To use this feature, before calling **cThrdInit()** to initialize the database engine, call **ctSetConfigurationOptions() (https://docs.faircom.com/doc/ctreeplus/ctsetconfigurationoptions.htm).**

For your convenience, we have implemented this functionality in **ctixmg**, a c-tree sample application.

Function pass FairCom DB license options in buffer

The c-tree Server DLL (shared library on Unix) supports a way of passing c-tree Server license options to the database engine rather than requiring a license file.To use this feature, before calling **cThrdInit()** to initialize the database engine, call **ctSetLicenseOptions() (https://docs.faircom.com/doc/ctreeplus/ctsetlicenseoptions.htm).**

For your convenience we have implemented this functionality in **ctixmg**, an example application built in the topic titled *Bound Server example - ctree.srvdll*.

4. SQL

Relational SQL access remains a powerful force for data reporting and analytics. FairCom DB allows legacy data to be linked to SQL. FairCom DB V12 makes this process automated on-the-fly. New SQL features such as multi-value sets and extended usage of parameters adds more flexibility. Extended JSON logging will greatly enhance query audit logging and diagnostics.

4.1 Dynamically "SQLize" ISAM and COBOL Files

Preserve Imported Data Files upon SQL DROP

FairCom DB SQL brings many advanced SQL capabilities to application data that was not originally created in SQL. Legacy application tables can be linked to SQL. FairCom DB SQL now defaults to always removing linked physical data and index files when performing a DROP from SQL as expected from SQL standards. However, there are frequently cases where removal of the data file is not appropriate nor expected. Rather, it is best to only remove the SQL system table linkage entry. A new configuration option is available to preserve the physical files.

```
SQL_OPTION DROP_TABLE_DICTIONARY_ONLY
```

Identify "Bad" Records from Legacy Data Linked to SQL

Multiple customers in production, in particular those using SQL callbacks, have encountered a situation where there was an error interpreting field content.

FairCom RTG offers a SQL syntax extension, `ctoption(badrec)`, which helps identify "bad" records and fields that cannot be properly interpreted. We extend this function into the general SQL engine itself to aid identifying "bad" records for all applications.

Example:

```
SELECT * FROM <table> ctoption(badrec)
```

When using the `ctoption(badrec)` syntax, only failing records are returned and error information is logged in *sqlserver.log*.

Auto Import Callback

FairCom has implemented the ability to "auto import" a table into SQL at the time it is accessed. This feature requires the user to provide a SQL callback library (*ctsqlcbk*) implementing the following:

1. The **ctsqlCallbackLoaded** function handling a call with `type == 2` setting `*ptr` to point to the function in 2. below.

2. A thread-safe function in *ctsqlcbk* that, given the database name, the table owner, and the table name, provides the necessary CTSQLIMPOPTS setting to be used to import the table, in particular the filename on disk.

   The function prototype is:

```
CTDBRET ctsqlImpSetOptsCallbackFunc (pCTDBSESSION pSession, pTEXT dbname, pTEXT tblowner, pTEXT tblname,
CTSQLIMPOPTS **opts);
```

where:

- *pSession* [IN] - the session handle
- *dbname* [IN] - the name of the database in use
- *tblowner* [IN] - the table owner of the table
- *tblname* [IN] - the name in SQL of the table
- *opts* [OUT] - pointer to a CTSQLIMPOPTS structure holding the information to internally call **ctSQLImport** function. The CTSQLIMPOPTS dbsnam, srvnam, usrnam, usrpwd, nonint members settings are ignored.

The SQL server engine at startup loads the *ctsqlcbk* and makes a call to **ctsqlCallbackLoaded** passing `type == 2` and registering the function returned in *ptr*.

At runtime, when the SQL parser encounters an object not present as a table in the system tables, if the registered function is not NULL and the user has resource permission or DBA permission when the table owner is someone else, the code calls the registered function and if the function returns CTDBRET_OK and sets the `*opts != NULL` attempts an automatic sqlimport using the *opts* returned overwriting the ignored members listed above as necessary for an internal server automatic sqlimport.

The "import" occurs in its own independent transaction (it cannot be reverted by rolling back the transaction). If the import fails for any reason, SQL statement execution continues as if the table simply does not exist.

After running the SQL callback function, a function is called to signal the memory allocated for *opts* should be released as follows:

```
ctsqlImpSetOptsCallbackFunc (pSession, NULL, NULL, NULL, opts).
```

See also *Auto Import: Tables that are missing on disk will now be Auto Purged (page 36).*

Auto Import: Tables that are missing on disk will now be Auto Purged

The FairCom DB SQL Auto Import now supports an "Auto Import" callback. When this new callback has been registered, the server should adjust the dictionary in the system tables to remove info for tables that are no longer on disk. The idea being we will constantly update our dictionary to match the files present on disk that have been opened. This means c-tree will automatically purge the information out of the system tables for tables that were previously auto imported but are now missing on disk, and add information for tables (Auto Import) that are opened but were not in the dictionary.

When a table is not found on disk where it was when added to the system table, it is purged and the error **-20005** is sent back to the application. No attempt is made to try to see if the table can be imported again.

Preserve Imported Data Files upon SQL DROP

FairCom DB SQL brings many advanced SQL capabilities to application data that was not originally created in SQL. Legacy application tables can be linked to SQL. FairCom DB SQL now defaults to always removing linked physical data and index files when performing a DROP from SQL as expected from SQL standards. However, there are frequently cases where removal of the data file is not appropriate nor expected. Rather, it is best to only remove the SQL system table linkage entry. A new configuration option is available to preserve the physical files.

```
SQL_OPTION DROP_TABLE_DICTIONARY_ONLY
```

Identify "Bad" Records from Legacy Data Linked to SQL

Multiple customers in production, in particular those using SQL callbacks, have encountered a situation where there was an error interpreting field content.

FairCom RTG offers a SQL syntax extension, `ctoption(badrec)`, which helps identify "bad" records and fields that cannot be properly interpreted. We extend this function into the general SQL engine itself to aid identifying "bad" records for all applications.

Example:

```
SELECT * FROM <table> ctoption(badrec)
```

When using the `ctoption(badrec)` syntax, only failing records are returned and error information is logged in *sqlserver.log*.

4.2 FairCom DB SQL Import and FairCom RTG Sqlize No Longer Require Exclusive Access to Tables

Historically, the FairCom DB SQL Import procedure (called "sqlize" in FairCom RTG) required exclusive access to the table being imported. This requirement has been relaxed so the SQL Import operations do not require opening tables in exclusive mode.

4.3 Run a SQL Query across Multiple Databases

FairCom DB SQL allows opening multiple local database from the same server. Databases are opened automatically when referenced in a SQL statement, and the same credentials are used for all databases. To access a table in a database other than the default, table name is qualified with a database name and a user name: *<dbname>.<owner>.<tablename>*. Tables in database other than the default database must be fully qualified, that is both the database name and the owner name must be specified.

Given two databases, the default "ctreeSQL" and "tutor_db". ctreeSQL contains a table "customer" and owned by the "admin" user. "tutor_db" contains a table named "customer" owned by user "tutor".

```
ISQL> select * from admin.customer, tutor_db.tutor.customer;
```

Trigger Execution

The execution of triggers takes place in the context of the table on which the trigger resides. Therefore, an insert into a table in the default database will execute a trigger on that table in the context of the default database. An insert into a table in a non-default database, will execute a trigger on that table in the context of the non-default database.

In the FairCom SQL Reference Guide, see *Cross Database Query* (https://docs.faircom.com/doc/*sqlref/sql-cross-database-query.htm*).

4.4 Parameter Marks Now Available in Scalar Functions and CASE Statements

FairCom DB now supports use of parameters in case expressions and in calls to scalar functions. Parameters are supported in most instances, with the exception of a few cases where it is not possible to determine the type of a parameter. Exception cases where parameters are not supported:

- TO_CHAR, first argument
- NULLIF, first argument
- COALESCE
- DATALENGTH

Example

```
ISQL> create table test2 (name char(10), item_count integer, invoice_date DATE);
ISQL> insert into test2 values ('hammer', 50, now() );
1 record inserted.
ISQL> commit;

ISQL> select * from test2 where ADD_MONTHS(invoice_date, ?) > SYSDATE;

Enter value for:
Parameter: 1
Type: INTEGER
Maximum size: 10
Is null (Y/N): n
Value: 6

NAME                 ITEM_COUNT INVOICE_DATE
----                 ---------- ------------
hammer               50 10/05/2020
1 record selected
```

4.5 Assign Values to Auto-Increment Fields in INSERT

SQL - auto_increment fields enhanced

IDENTITY column support is now "smart" and automatically inserts values when explicitly specified and auto-generates values when they are not (the default existing behavior).

The syntax/definition of `auto_increment` fields in this release is identical to the current `IDENTITY` syntax except `INSERT` statements are smarter. When an `INSERT` statement specifies a value for an `IDENTITY` column, it inserts the specified value; when the value is omitted, it is generated automatically.

Prior to this modification, it was necessary to explicitly disable IDENTITY insertion with `SET identity_insert <table> on | off`.

Only one table at a time can be set to identity_insert on.

No table definition changes are required to enable this new support. It enables compatibility with many external SQL frameworks such as SQLAlchemy.

4.6 Insert Multiple Value Sets

Multiple rows can now be inserted with a single INSERT statement using multiple value sets.

Note: Use of parameters is not supported with multiple value sets.

Example

```
ISQL>create table mult (f1 int, f2 int);
ISQL>insert into mult values (1,1),(2,2),(3,3);
3 records inserted.
ISQL> select * from mult;
F1 F2
-- --
1 1
2 2
3 3
3 records selected
```

4.7 Insert Statements with Scalar Values and Subqueries Now Supported

FairCom DB SQL has been enhanced to handle mixing scalar values and a subquery in an INSERT INTO statement. A script similar to the following now succeeds:

```
create table source (f1 char(10));
insert into source values ('aaa');
create table dest (f1 integer, f2 char(10));
insert into dest values(1, (select f1 from source));
```

4.8 Use Row Value Constructors with Comparisons in Query

Use row value constructors for comparisons in query statements.

Example

```
ISQL> table rvc
COLNAME NULL ? TYPE LENGTH CHARSET NAME COLLATION
------- ------ --- ------ ------------ ---------
c1 INT 4
c2 INT 4
c3 INT 4
ISQL> select * from rvc;
C1 C2 C3
-- -- --
1 1 1
1 1 2
1 1 3
1 2 1
1 2 2
1 2 3
1 3 1
1 3 2
1 3 3
2 1 1
2 1 2
2 1 3
2 2 1
2 2 2
2 2 3
2 3 1
2 3 2
2 3 3
3 1 1
3 1 2
3 1 3
3 2 1
3 2 2
3 2 3
3 3 1
3 3 2
3 3 3
27 records selected
ISQL> create index rvc_idx on rvc(c1,c2,c3);
ISQL> select * from rvc where c1 > 2 and c2 > 2 and c3 > 2;
C1 C2 C3
-- -- --
3 3 3
1 record selected
```

```
ISQL> select * from rvc where (c1,c2,c3) > (2,2,2);
C1 C2 C3
-- -- --
2 2 3
2 3 1
2 3 2
2 3 3
2 3 1
2 3 2
2 3 3
3 1 1
3 1 2
3 1 3
3 2 1
3 2 2
3 2 3
3 3 1
3 3 2
3 3 3
13 records selected
select * from rvc where c1 < 2 and c2 < 2 and c3 < 2;
C1 C2 C3
-- -- --
1 1 1
1 record selected
select * from rvc where (c1,c2,c3) < (2,2,2);
C1 C2 C3
-- -- --
1 1 1
1 1 2
1 1 3
1 2 1
1 2 2
1 2 3
1 3 1
1 3 2
1 3 3
2 1 1
2 1 2
2 1 3
2 2 1
13 records selected
select * from rvc where c1 <= 2 and c2 <=2 and c3 <= 2;
C1 C2 C3
-- -- --
1 1 1
1 1 2
1 2 1
1 2 2
2 1 1
2 1 2
2 2 1
```

```
2 2 2
8 records selected
select * from rvc where (c1,c2,c3) <= (2,2,2);
C1 C2 C3
-- -- --
1 1 1
1 1 2
1 1 3
1 2 1
1 2 2
1 2 3
1 3 1
1 3 2
1 3 3
2 1 1
2 1 2
2 1 3
2 2 1
2 2 2
14 records selected
```

4.9 Easier Full-Text Search (FTS) MATCH Operator Syntax

To perform a phrase match, the phrase must be enclosed in single quotation marks per FTS query syntax defined at the c-treeDB level. In SQL, that mandates two single quotes since the entire query is surrounded by a single quote (because it is a literal). Therefore, quotes need to be escaped in the standard way (that is double them).

(This implies the entire query is a phrase match since the final query is made by 3 single quotes, the phrase and 3 single quotes.)

```
SELECT * FROM docs WHERE document MATCH '(''faircom'' AND ''database'')'
```

For convenience, the FairCom DB SQL syntax now allows wrapping phrases within the FTS query literal with double quotes (that do not need to be escaped).

```
SELECT * FROM docs WHERE document MATCH '(“faircom” AND “database”)'
```

Both the old and the new methods (double quote or double single quotes) of specifying phrase search can be used for SQL FTS queries.

4.10 Diagnostic Logging Now in Enhanced JSON Format

SQL statement logging output has been revised to a single-line JSON format and includes the timestamp, user, and IP-address of the client making the request. This new format should allow easier ingestion of these logs into external monitoring systems. The following configuration options now output in this format:

```
SQL_DEBUG LOG_STMT
SQL_DEBUG LOG_STMT_TIMES
SQL_DEBUG LOG_STMT_TIMES_FETCH
```

Example Output

```
{"timestamp":"Fri Aug 24 11:56:24
2020","ipaddr":"::ffff:127.0.0.1","db":"CTREESQL","user":"admin","thread":23,"operation":"OPEN","exect
ime":7,"tmptime":2,"statement":"select * from syscolumns order by width "}
```

The `SQL_DEBUG LOG_STMT_TIMES` format has been changed as follows:

1. It no longer logs statement times of the PREPARE operation.
2. The time precision is now in milliseconds.

In the case of a parameterized string, `SQL_DEBUG LOG_STMT_TIMES` does not log the parameter value anymore.

4.11 Extensive SQL Statement Logging for Auditing

SQL Statement logging can be an essential DBA tool for identifying performance issues and unexpected queries. `SQL_DEBUG LOG_STMT` adds extensive statement logging in *sqlserver.log*. This information includes connection information, improved timing, and logging the statement before it is actually executed for a detailed audit trail of all SQL operations.

The main advantage compared to `SQL_DEBUG_LOG_STUB_MED` is that the logging allows identifying which connection sent the statement so that it is easier to understand the statement executed by a specific connection.

Compared to the `SQL_DEBUG LOG_STMT_TIMES` keyword, this new keyword generates less output and the statement is logged before execution (`LOG_STMT_TIMES` log after execution). In case of a very long running query, this makes it possible to identify the statement by looking at the log.

4.12 SYSLOG SQL_STATEMENTS Configuration Keyword

This configuration keyword logs executed SQL statements in *SYSLOG*:

```
SYSLOG SQL_STATEMENTS
```

A `SYSLOG SQL_STATEMENTS` (https://docs.faircom.com/doc/ctserver/syslog-config.htm) log entry is written
after statement execution so it can also include the error code (if any).

The variable part of the `SYSLOG` entry contains statement information in JSON format similar to `SQL_DEBUG LOG_STMT`. (https://docs.faircom.com/doc/sqlops/sql-debug-log-stmt-config.htm)

Below is a sample showing how it is displayed by the **ctalog** utility:

```
Class           = 16 (SQL)
Event           = 1 (SQL statement)
Date            = 09/24/2020
Time            = 17:40:11
Sequence number = 37
Error code      = -20005
User ID         = 'admin'
Node name       = 'isql'
Variable-length information:
-------------------------------------------------
{"timestamp":"Tue Sep 24 17:40:27
2020","ipaddr":"127.0.0.1","db":"CTREESQL","user":"admin","thread":29,"statement":"select * from
missingtable"}
-------------------------------------------------
```

4.13 LVARCHAR Fields Allowed in Stored Procedure Code

Within a Java stored procedure, it is now possible to retrieve, insert, and update the content of an LVARCHAR field. the following changes have been made:

SQLIStatement and **SQLPStatement** methods have been enhanced such that the **SetParam** method can be used to specify the content of an LVARCHAR field.

The **SQLCursor GetValue** method has been enhanced to retrieve the entire content of an LVARCHAR field.

The SQLCursor class has a new member getLongValue to retrieve a chunk of a LVARCHAR field (in case the field content is too large).

Function

```
public Object getLongValue(int field, int offset, int len)
```

where:

- *field* - the field number
- *offset* - the starting point (0 based) where the portion you want to retrieve starts.
- *len* - the length of the portion to retrieve.

Return

The function return NULL if there is no more to retrieve.

4.14 Throw Custom Error Message on Stored Procedure or UDF Exception

This modification provides the capability to return an error message from a stored procedure or user-defined function (UDF).

Before this modification, the only way for a stored procedure to set the return code to a value different than 0 was to throw a **DhSQLException()** either directly (only when using the NetBeans plugin to generate the stored procedure) by:

```
throw new DhSQLException(678,"my error message");
```

or through the DhSQLThrow class:

```
DhSQLThrow.throwSQLException(678,"my error message");
```

In both cases the error code returned was interpreted as a known SQL error code and the error message was set to the one for the specific error code returned, ignoring the message.

This modification adds a preferred way of returning an error message by calling the new **fail(***String message***)** method. This method causes the stored procedure to terminate with error **-26015** and sets the error message to the string passed in. The message will be truncated if it is larger than 510 characters. For example:

```
ISQL> CREATE PROCEDURE  excp()
BEGIN
fail("my error message");
END
ISQL> call excp();
ISQL> error(-26015): my error message
```

4.15 Dynamically Disable Triggers

When a table contains a trigger, it can be desirable to avoid firing that trigger when performing administrative tasks such as bulk updates. This modification adds the possibility to disable triggers for the current session by calling:

```
SET TRIGGER OFF
```

Trigger executions can be reestablished by

```
SET TRIGGER ON
```

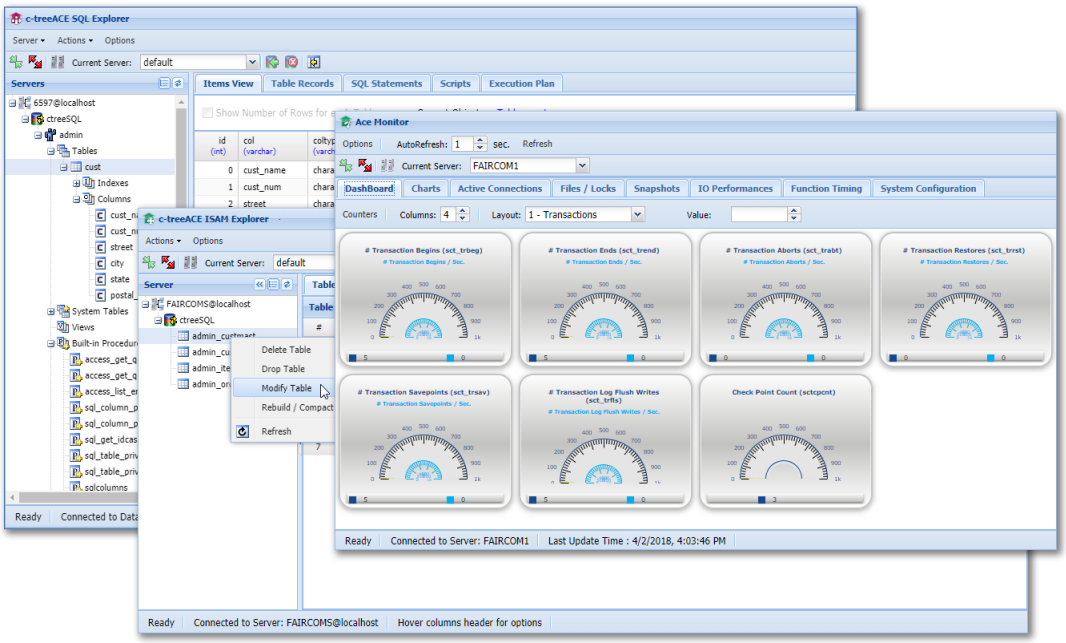
The user executing these commands needs to have resource permissions on the database.

5. Goal: Zero Administration

FairCom DB strives to be "DBA Free" for easy deployment and low TCO (total cost of ownership). New tools and automation of tasks such as log pruning, data purging and automatic timestamps are all new options included in this release.

5.1 Browser-Based Web Tools

Browse and manage your c-tree Server environment from nearly anywhere on the network directly from your browser! V12 makes available a suite of browser-based utilities featuring full-functioned visual user interfaces for monitoring operations and exploring your FairCom DB databases.



Periodic updating of these tools is also now easier than ever keeping your entire organization up-to-date. All components are centrally located on the server via a web service. Simply replace components when provided and all users will immediately enjoy the latest utility updates and features. No more desktop application version and installation hassles.

Details on these tools are available in *FairCom Graphical Tools* (https://docs.faircom.com/doc/*faircom-graphical-tools/*).

5.2 Automatic System Timestamps

The FairCom Database Engine supports multiple data automation features to quickly and efficiently process data streams. These features can be used independently or combined for optimizing data capture.

- Automatically stamp data records with the current date and time as they are captured.
- Automatically aggregate selected columns for advanced data analytics.
- Automatically purge data as it ages to minimize storage retention on devices with limited storage capacity.

These features are ideal for quickly capturing streams of data, automatically timestamping the data, automatically aggregating the data, and automatically retaining desired data and purging old data.

You have extensive declarative control over each of these features, which makes it easy to enable or disable timestamping, or control the time period before purging data, or controlling aggregation time windows for average, min, max, etc.

Full details are available here: *Automatic Timestamps, Aggregation, and Purging* (https://docs.faircom.com/doc/*aggregation/*).

Other enhancements available in this release that aren't currently in the Automatic Data Aggregation book are discussed below.

Hot Alter Table Automatic System Timestamp

Support for **ctdbAlterTable()** has been added to include Automatic System Timestamp features. This function can be used to:

1. Set/reset an existing field to be auto system time
2. Change the auto system time feature of a field
3. Add a new field with auto system time
4. Delete a field with auto system time

During Alter Table, when setting/resetting auto system time, the field content is not modified, i.e., the current timestamp is not applied during Alter Table to existing record contents.

New functions for automatic system timestamp support

In V12 and later, the following c-treeDB functions have been added for use with columns that support Automatic System Timestamp:

- **ctdbSetFieldAutoSysTime** (https://docs.faircom.com/doc/**ctreedb/ctdbsetfieldautosystime.htm**)
- **ctdbGetFieldAutoSysTime** (https://docs.faircom.com/doc/**ctreedb/ctdbgetfieldautosystime.htm**)

Automatic system time field definition

As part of the support for automatic timestamps, the following new functions have been added to c-tree's API in V12 and later:

- **AddAutoSysTimeFields()** (https://docs.faircom.com/doc/**ctreeplus/addautosystimefields.htm**)
- **WhichAutoSysTimeFields()** (https://docs.faircom.com/doc/**ctreeplus/whichautosystimefields.htm**)
- **RemoveAutoSysTimeFields()** (https://docs.faircom.com/doc/**ctreeplus/removeautosystimefields.htm**)
- **UpdateAutoSysTimeFields()** (https://docs.faircom.com/doc/**ctreeplus/updateautosystimefields.htm**)

High-resolution timestamp support

Two c-treeDB API functions have been added for high-resolution timestamp generation:

- **ctdbCurrentTimestamp()**
(https://docs.faircom.com/doc**/ctreedb/ctdbcurrenttimestamp.htm**)
- **ctdbCurrentDateTimeUTC()**
(https://docs.faircom.com/doc**/ctreedb/ctdbcurrentdatetimeutc.htm**)

5.3 Automatic Data Aggregation

Automatic Data Aggregation simplifies the task of performing simple processing on timestamped data. For example, a continuous series of readings from a temperature sensor can be stored in a table. The Automatic System Time function is used to add data timestamps if necessary. Automatic Data Aggregation is used to calculate average temperature readings based on this data.

Some applications, especially those in the Internet of Things, can generate a huge amount of data due to the nature of sensors, many of which produce a constant stream of readings. This may result in more data than can be stored locally on a small device. It may be impractical to send all of the raw data to the cloud for storage or processing due to bandwidth considerations. Developers may be interested in storing data that has been aggregated over a given amount of time. For example, a temperature sensor may produce readings many times in a minute, however the moving window average of those readings may be more meaningful than individual data points and requires far less storage.

Automatic Data Aggregation provides developers and DBAs an easy solution for aggregating and storing data.

Automatic Data Aggregation monitors Automatic System Time fields, aggregates data record values within a given amount of time (user specified sample window), and stores defined aggregated values in an aggregation table.

How to Use Automatic Data Aggregation

This plug-in feature is enabled by adding the following server configuration in *ctsrvr.cfg*:

```
PLUGIN cttimestamp;<cttimestamp dynamic library>
```

where:

- *<cttimestamp dynamic library>* is the path and filename of the plug-in, e.g., *server\aggregation\ctTimeStamp.dll*.

This plug-in is loaded at server startup and begins aggregating tables listed in the *cttimestamp.json* configuration file described below.

Requirements:

- The table to aggregate must have at least one auto-timestamp field of type CT_TIMESTAMP.
- The auto-timestamp field must be the *first* segment of any index on that table.

Supported Field Types - The fields to aggregate must have one of the following field types:

- Signed integer types: CT_TINYINT, CT_SMALLINT, CT_INTEGER, CT_BIGINT

- Unsigned integer types: CT_UTINYINT, CT_USMALLINT, CT_UINTEGER, CT_UBIGINT
- Float types: CT_SFLOAT, CT_EFLOAT, CT_DFLOAT

5.4 Automatic Data Purging

The Automatic Data Aggregation family of features simplify the collection and aggregation of sensor data. Many real-time sensors generate large streams of data, which can quickly fill available storage on smaller devices. The Auto-Purge feature can be used to purge the oldest data making room for new data. Before purging, the data can be sent to the cloud for storage — depending on your environment, either the entire set of raw data can be streamed to the cloud or, if bandwidth is a consideration, a smaller set of aggregated values can be sent.

c-tree leverages the advanced partitioning capabilities of the FairCom Database Engine to accomplish this. Partitioning allows you to divide a database into smaller partitioned files. As records are inserted into the database, they are sorted into the partitions based on defined criteria, such as time or date. For example, for sensors producing a large number of records, you can set a partition to accommodate an hour's worth of records: Each hour, a new partition is created to hold that hour's records.

The Auto-Purge feature allows automatic purging of data so the storage available on your device does not fill. Using this feature, you might purge aged data so that, say, only 24 hours of current data are kept on the device. In the example of setting each partition to hold data for 1 hour, 24 individual file partitions are created on the device. As soon as the 25th partition is begun, the 1st partition is purged. When the 26th partition is begun, the 2nd partition is purged. And so on. The data collection demands of your application determine how much data is stored in a partition and how many partitions to keep.

In the case of "time series" data, where a new record is constantly inserted into the table (such as a stream of sensor data), new partitions will be created constantly and Auto-Purge will be constantly removing old partitions. *Notice that, if you stop inserting records, the old partitions will never be removed.*

API Support

Support for partitions with Auto-Purge is included in the FairCom DB APIs. Typically, you only need to add a REST call parameter.

If you are developing with the other APIs, consider the topics in this section:

- *REST API Support for Auto-Purge* (page 89)
- *FairCom DB C++ Support for Partitions and Auto-Purge* (https://docs.faircom.com/doc/*ctreepp/*)
- *c-treeDB Support for Auto-Purge* (https://docs.faircom.com/doc/*ctreedb/ctdbsettablepartitionmaximumactive.htm*)
- *FairCom Low-Level API Support for Auto-Purge* (https://docs.faircom.com/doc/*ctreeplus/maxautopurgepartitions.htm*)

5.5 Automatically Alert on Low Disk Space

Storage space continues to grow. However, it never seems just quite enough. FairCom DB requires enough space available to ensure data is secured, and this is a strict requirement for transaction controlled files. Running out of storage space will force FairCom DB to immediately terminate operations, which is a real risk to critical business operations.

FairCom DB introduced a background thread (https://docs.faircom.com/doc/ctserver/subsystem-disk-full-action-config.htm) to monitor storage space with administration alerts at defined thresholds. To make this easier to find and implement we have included this configuration commented ready to go in the default *ctsrvr.cfg*.

```
; Disk Mull Monitoring
;SUBSYSTEM EVENT DISK_FULL_ACTION {
     volume          VOLUME
     limit           LIMIT
     run             EXE [OPTIONS]
     freq            FREQUENCY
     maxruntime      MAXRUNTIME
}
```

This configuration allows you to specify a volume to monitor, how often to check and what to run when the limit threshold it crossed. The run script can execute anything in the path of FairCom DB. For example, send an email or log a message to an external monitoring system.

To enable this, uncomment the first line and set your configurations and restart. And, of course, you can have multiple of these onions in place for monitoring multiple volumes.

5.6 Automatic Sizing and Purging of Log Files

Logging is a critical background task that must be done. When exceptional events happen you expect to immediately trace them through available log files. A challenge with logs is that they grow, and sometimes unbounded. This impacts storage space, which must not be completely consumed.

FairCom DB provides options to keep logs within configurable sizes and purge log data as it rolls over. Two central logs can be managed in this regard.

CTSTATUS.FCS

CTSTATUS.FCS is your "go to" log for all things FairCom DB. Any exceptional event encountered by server operation is logged to this text based file. It is critical to know the immediate locaiton of this file at all times. Any time you request FairCom support, we will likely ask for this file as it contains valuable information about the running parameters of your server. This file can grow very large in some environments. For example, when logging dynamic backup events for every file, this file gets exceptionally large.

To limit the size of this file, `CTSTATUS_SIZE` is included as a default configured size of 32MB.

sql_server.log

sql_server.log maintains log information specifically for SQL operations. This can be dynamically enabled when needed for statement audit logging or logging stored procedure operations. For example `SQL_DEBUG LOG_STMT` (https://docs.faircom.com/doc/`sqlops/sql-debug-log-stmt-config.htm`)
enables statement logging.

Now, you can limit the size of *sql_server.log*. When it fills to the configured limit, a new file is created, the current log is marked as *sql_server.bak* and the prior backup purged. That is, one active and one backup log is always maintained. Configure your *sql_server.log* with the `SQL_SERVER_LOG_SIZE`
(https://docs.faircom.com/doc/`sqlops/SQL_SERVER_LOG_SIZE.htm`) configuration.

6. Diagnose Easier

New monitoring options allow easier diagnostic checks for performance and reliability. Over 400 metrics are available for reporting on every aspect of FairCom DB operation including cache buffers, transaction control, file I/O, user connections, memory usage, and many many more. Utilities (https://docs.faircom.com/doc/ctreeplus/ctstat-util.htm) reporting these statistics have been updated in parallel for immediate usage.

6.1 Track I/O Statistics per Connection

Resource consumption is a closely watched metric in database usage. Tracking notable I/O usage to a specific user or application connection is a common performance profiling task. FairCom DB now provides options for I/O monitoring per connection.

A new **ctstat -userinfox option** provides the following statistics for each connection:

- disk read bytes
- disk write operations
- disk write bytes
- data cache requests
- data cache hits
- index cache requests
- index cache hits

See Also

This feature is backed by a new FairCom DB API function that can be embedded directly into any application monitoring module. See **USERINFOX() (page 62) in the programming guide for details.**

6.2 File Operations Counters

Database resource consumption requires close monitoring of all file I/O activities. FairCom DB tracks values for additional physical file operations. The following values are tracked:

- logical file opens
- logical file closes
- physical file opens
- physical file closes
- file creates

- file renames
- file deletes

Each value is a cumulative value since server startup, stored as an 8-byte integer field in the system snapshot structure.

The counters include every call, not just successful calls. Note that the physical file open count can be smaller than the physical file close count, because when a file is created it does not increment the physical file open count.

The **ctstat** utility supports a new option, *-fileops*, that displays these counters. Example:

```
ctstat -fileops -t -i 1 -h 10 -u ADMIN -p ADMIN -s FAIRCOMS

  logopn/s  logcls/s  phyopn/s  phycls/s  filcre/s  filren/s  fildel/s  total/s
         0         0         0         0         0         0         0        0
         0         0         0         0         0         0         0        0
       154       158        23        26        13         0         3      377
       959       926        70        63        25         0         4     2047
      1016       989         4         0         0         1         1     2011
       574       562         0         0         0         0         0     1136
       481       486         2         0         0         0         0      969
       425       417         0         0         0         0         0      842
       376       380         0         0         0         0         0      756
```

The **ctstat** *-fileops* option requires a server that uses snapshot version 21 or later. If this option is used with a server that uses an earlier version of the utility, the utility fails with the error message:

```
Error: The -fileops option requires snapshot version 21, but this server is using snapshot version <version>.
```

where `<version>` is the older snapshot version that FairCom DB is using.

The snapshot file parsing utility, **ctsnpr**, has been updated to support the new *SNAPSHOT.FCS* file format.

6.3 Debug Heap Options for Detection of Memory Corruption

FairCom Database Engine heavily utilizes memory operations to optimize performance. To avoid frequent OS memory requests, it maintains a sophisticated memory suballocator. This internal subsystem allocates larger blocks of memory from the O/S and sub allocates them as needed. All memory gets and puts from this suballocator system are tagged, tracked and checked. On rare occasions, heap memory corruption can occur resulting in an eventual illegal memory access attempt, ultimately terminating FairCom DB operations. Debugging these conditions is extremely complex. FairCom DB V12 and FairCom RTG V3 enhance memory diagnostics when required.

Debugging options can now be enabled when corrupted memory is suspected triggering additional tracking and checking of internal memory suballocations. Use of these options should

only be considered in consultation with your FairCom support team as these can significantly impact performance in certain use cases.

`HEAP_DEBUG_LEVEL <level>` with values 0-3 (Default 0):

- `HEAP_DEBUG_LEVEL 0` uses the default memory allocator; no checks are performed.
- `HEAP_DEBUG_LEVEL 1` enables the debug allocator, with some detection for memory buffer write overruns or use-after-free bugs. There is no additional memory overhead, and the CPU overhead is small.
- `HEAP_DEBUG_LEVEL 2` does the checks from `HEAP_DEBUG_LEVEL 1` and adds a small redzone before/after each allocation, which is checked at free. This may detect some buffer write overruns, write underruns, or double frees. This option uses an additional 16 bytes of memory on each allocation.

  Levels 1 and 2 generate a *CTSTATUS* entry and stack trace at free when a memory fault is detected.
- `HEAP_DEBUG_LEVEL 3` uses the system's virtual memory subsystem to immediately detect illegal memory overruns (both reads and writes) by aligning allocations at the end of a 4K page. The following types of errors may immediately generate a segmentation fault (core dump) on the invalid access: memory read/write overrun, double free, and use-after-free.

  Memory write underruns or some small write overruns may instead be detected at free and generate a *CTSTATUS* message and stack trace.

  Memory allocation failures are logged to *CTSTATUS* logging when `HEAP_DEBUG_LEVEL 3` is enabled.

  Because Level 3 debugging uses an extra 4K-8K bytes for all allocations, it is possible to restrict this to particular size(s) of allocations using `HEAP_DEBUG_EXCLUSION_LIST <N>`.

  Be sure to see the **Performance Note** below.

`HEAP_DEBUG_EXCLUSION_LIST <N> [,<N> ...]` - When `HEAP_DEBUG_LEVEL 3` (only) is specified, it is possible to have particular allocation sizes use the default (non-debug) allocator to reduce the overall memory overhead of this debugging method. By default, `HEAP_DEBUG_LEVEL` applies to all allocation sizes. The particular values to specify would typically be recommended by FairCom support based on analysis of prior core dumps.

```
HEAP_DEBUG_EXCLUSION_LIST # allocation size range (in bytes):
#  Bytes
1  <=16
2  17-32
3  33-64
4  65-128
5  129-256
6  257-512
7  513-1024
8  1025-2048
9  2049-4096
10 4097-8192
```

Limitations: When Level 3 debugging is enabled, memory usage statistics are not tracked for those bins.

Performance Note

`HEAP_DEBUG_LEVEL 3` has a negative impact on performance on any machines with 4 or more CPU cores. FairCom does not recommend the use of Level 3 on any machines with 4 or more CPU cores where performance is important.

If performance is an issue, FairCom recommends using the Level 2 version of heap debug on any location where you've seen a memory-related crash occur and where performance is important. Our testing has not revealed a noticeable performance impact with Level 2, even on a large test box with 72 cores.

Note 1:

If allocation stack traces are enabled (such as with **ctstat** *-mt +ALL*), an additional stack is dumped showing the allocation location of the buffer.

A use-after-free might cause a segmentation fault (core dump).

Note 2:

`HEAP_DEBUG_LEVEL 2` has a benefit to stability: If the buffer overrun is 8 bytes or less, the heap will NOT be corrupted because only the extra redzone memory is modified. The server will stay up and only a stack trace will be generated.

Linux

On Linux, the `HEAP_DEBUG_LEVEL 3` configuration option requires raising the kernel limit `vm.max_map_count` to be twice the number of allocations. FairCom can help you to determine the minimum value to set the Linux kernel parameter, `vm.max_map_count`. (This value will generally rise and fall with the amount of server activity, so include a safety factor based on how much busier it might be.) Setting `vm.max_map_count` too small will result in allocation failures if the number of allocations exceeds that limit, which could lead to a server crash.

This can be changed until next system reboot with the command:

```
sysctl -w vm.max_map_count=<N>
```

Example using HEAP_DEBUG_LEVEL 3 restricted to sublist #2:

Look at current memory usage at a busy server time using this command:

```
ctstat -ml -u admin -p ADMIN_PASSWD -s FAIRCOMS
```

The output will show sublist #2 as "`PI2TYP`." The next column shows the total allocations in bytes.

```
PI2TYP 98304 7400 0.01%
```

Divide the first column by `32` (the max size of this allocation range) to get the number of current allocations on this sublist: `98304/32=3072`, and then double that: `6144`.

You also need to include the `MBATYP`:

```
MBATYP 73388776- 73388776-
```

Divide the first column by `8192` (this would be the worst case assumption for this list), and double the result:

```
73388776/8192 * 2 = 17916
```

The sum of these values is `2x` the allocation count: `17916+6144 = 24060`. Apply a safety multiplier such as `x10`.

This is the minimum value you need to set as the Linux kernel parameter:

```
vm.max_map_count
```

This value will generally rise and fall with the amount of server activity, so include a safety factor based on how much busier it may become. Setting `vm.max_map_count` too small will result in allocation failures if the number of allocations exceeds that limit, which could lead to a server crash.

After setting the kernel value (Linux only) with `vm.max_map_count`, add the following server keywords to enable the debug suballocator on sublist #2 (`PI2TYP`) only:

```
HEAP_DEBUG_LEVEL 3
HEAP_DEBUG_EXCLUSION_LIST 1,3,4,5,6,7,8,9,10
```

Stack Dump Message

When a Stack Dump is generated, the following message will be logged to CTSTATUS.FCS.

```
"A Heap Fault was detected and stack dump created"
```

If you have this logic enabled, FairCom recommends routine reviews of *CTSTATUS.FCS* to determine if a Stack Dump has been created. If you see this message, please send the Stack Dump, and the *CTSTATUS.FCS* message to FairCom Support for inspection.

7. Backup and Restore

Backup and recovery are fundamental operations of a database. FairCom DB provides a robust full-feature ability to hot backup (https://docs.faircom.com/doc/ctserver/backups-and-data-integrity.htm) files at any time. Recovery is just as robust with point-in-time recovery and ability to extract only files of interest.
The latest release adds new options to extend and enhance database backup functionality including methods for compression and encryption.

7.1 Back Up Direct to STDOUT and Gain OS Compression and Encryption Support (ctdump)

Database backups are essential, and should be taken at every opportunity and at regular intervals. However, backups are vulnerable to inappropriate access and can consume additional storage space. Encrypting and/or compressing these critical data streams is more important than ever.

The FairCom Database Engine can now redirect backups (dynamic dumps) to an operating system's standard output (*stdout*) channel. This makes it easy to use operating system utilities and third-party tools to process backups before you write them to disk. Any file utility that can read and write data from *stdin* and *stdout* can take advantage of this feature. In addition, chaining several processes together is very easy. This greatly reduces post-processing steps such that your backups are ready for immediate archiving. For example, you can pipe a backup into **gzip** to compress the stream, pipe results into **ccrypt** and encrypt them, and finally pipe the resulting file directly to FTP and transfer to another environment.

The **ctdump** (https://docs.faircom.com/doc**/ctserver/ctdump-util.htm**) utility has a new option (*-x*) that redirects a backup stream to *stdout* and sends error and informational messages to *stderr*. You must remove any
processing you applied before using the **ctdump** utility to restore these backups.

You must remove any processing in reverse order you applied before using the **ctdump (https://docs.faircom.com/doc/ctserver/ctdump-util.htm)** utility to restore these backups. *And don't lose or forget your*
encryption keys!

See **ctdump (https://docs.faircom.com/doc/ctserver/ctdump-util.htm)**

Restore Backups Direct from STDIN (ctrdmp)

The **ctrdmp** utility now supports the ability to restore a dynamic dump stream being redirected to standard input. To enable this behavior, **ctrdmp** must be run with the *-x* option, in addition to providing the proper redirection.

7.2 Restore Backups Direct from STDIN (ctrdmp)

The **ctrdmp** utility now supports the ability to restore a dynamic dump stream being redirected to standard input. To enable this behavior, **ctrdmp** must be run with the *-x* option, in addition to providing the proper redirection.

7.3 Wildcards Exclude and Include Files in Backups

The backup script already supports wildcard file names to easily specify a set of files to be backed up. A frequent situation is having a directory of many files and wanting to exclude several files from a backup. For example, some files are transient in nature and are frequently re-created with no need to be included in a backup. It was requested if this ability could be easier managed with file exclusion capability.

V11.9 and later now provides `!EXCLUDE_FILES`. This option excludes files from the backup that match wildcard specifications.

- `!EXCLUDE_FILES` must be specified before the `!FILES` section in the dump script.
- `!EXCLUDE_FILES` also applies to the `!NONCTREEFILES` list of files.

Example

Back up all files matching **.dat* and **.idx* except for files that match *test.**.

Reminder: `!FILES` and `!EXCLUDE_FILES` must go on a line all by themselves

```
!DUMP backup.fcb
!EXCLUDE_FILES
test.*
!FILES
*.dat
!END
```

7.4 Dynamic Dump Script !DELAY Option Allows Abandoning Dump

When the Dynamic Dump begins, by default it prevents new transactions from beginning and waits indefinitely for open transactions to complete. Once no transactions are active, a dump checkpoint is created, the dump creation begins, and transactions are allowed to resume. If the dump script option `!DELAY <N>` is specified, the Dynamic Dump waits up to `N` seconds for active transactions to complete before terminating them. This revision allows `!DELAY <N>` with `N < 0` to cause the dump be abandoned with error **DUMP_ABANDONED_ERR** (1162) if active transactions still remain after `|N|` seconds.

7.5 Faster Restores from Large Backups

Automatic Recovery could take a long time to complete if many files were referenced in the transaction logs. Backup recovery logic has been enhanced to greatly speed this process.

As an example, this performance enhancement reduced the time of **ctrdmp** restoring 1500 data and 1500 index files from 406 sec. to 63 sec.

Slow performance was sometimes experienced when using the **ctrdmp** (https://docs.faircom.com/doc**/ctserver/ctrdmp-util.htm**) utility to restore a large number of files and transaction activity occurred during the dynamic dump. Recovery utility performance has been improved for these activities. This change has no impact on configuration or compatibility. Using the updated **ctrdmp** (https://docs.faircom.com/doc**/ctserver/ctrdmp-util.htm**) utility resolves the slow performance for a dynamic dump
stream file that exhibited slow performance under these conditions.

8. High Availability (Beta)

FairCom is pleased to announce advancements in building Highly Available solutions with FairCom Technology. Full details on building High Availability and Disaster Recovery solutions are available in the FairCom Clustering documentation: *Clustering for Scalability and Availability* (https://docs.faircom.com/doc/*cluster-for-scalability-availability/*)

8.1 Synchronous Replication for High Availability

FairCom Server V12 and later support both asynchronous and synchronous replication for c-tree data files under full transaction control. The new Synchronous Replication Agent can better address cases where high availability is crucial:

- A failover environment, where the secondary (target) server must be always in sync with the primary (source) server.
- Cases where no committed transaction can be lost while moving from the primary to the secondary server.
- Cases where no delays are acceptable while moving to the secondary server.

With Replication Manager, the user can choose between synchronous and asynchronous replication when setting up replication plans.

Note: Depending on the synchronous replication mode selected, it is possible update operations may experience performance penalties.

Requirements

Synchronous replication requires files to have replication support enabled, which means they must meet the same requirements for data replication as previous asynchronous replication did:

1. the file must have a unique index
2. the file must be under full transaction control
3. the file must be marked for replication

Enabling Synchronous Replication with Replication Manager

When configuring replication, the Replication Manager browser-based application has a check box that allows a replication mode to be enabled as either Synchronous or Asynchronous. The replication mode can be set on a per-plan basis.

Enabling Synchronous Replication with Replication Agent

To enable synchronous replication for a file, follow these steps:

1. Set up a Synchronous Replication Agent. This involves creating a Replication Agent configuration file that includes the option `syncagent yes` and using the **repadm** utility to register the unique ID of the Replication Agent with the source server. For example, if the source server name is `FAIRCOMS@source_hostname`, and the Replication Agent unique ID

is AGENT1, the following command registers this agent ID with this source server (note that replication agent IDs are not case sensitive):

```
C:\> repadm -c addsyncagent agent1 -s FAIRCOMS@source_hostname
FairCom DB(tm) Version 12.0.81 (Build-201020) Replication Agent
Management Utility
Copyright (C) 1992 - 2020 FairCom Corporation
ALL RIGHTS RESERVED.
Successfully added synchronous replication agent 'agent1'
```

2. Use the **repadm** utility to enable synchronous replication for the data file on the source server. Note that the data file must exist for this operation to succeed. Example:

   ```
   C:\> repadm -c addsyncagentfilemap mark.dat agent1 -s FAIRCOMS
   FairCom DB(tm) Version 12.0.81 (Build-201020) Replication Agent
   Management Utility
   Copyright (C) 1992 - 2020 FairCom Corporation
   ALL RIGHTS RESERVED.
   Successfully added mapping to synchronous replication agent 'agent1' for
   file 'mark.dat'
   ```

   It is possible to enable synchronous replication for a data file on more than one secondary server. Simply create a synchronous replication agent for each secondary server and repeat the above steps for each replication agent.

   After these steps have been successfully completed, the commit of any transaction that updates the specified file will wait for notification from the replication agent(s) that the transaction has been processed.

3. Add the following option to the Replication Agent configuration file, *ctreplagent.cfg*, and start the replication agent:

   ```
   syncagent yes
   ```

   When this option is specified, the Replication Agent operates as a synchronous agent, sending notification of its commit position to the primary server.

Note that for a Replication Agent to successfully connect to a source server as a synchronous agent, its unique ID must have already been registered with the source server. Otherwise, the connection fails with error **101** and the following messages are logged to *ctreplagent.log*:

```
Failed to update commit position on source server: 101
Check that this replication agent's ID (<agentid>) is registered as a synchronous replication agent on the
source server
Synchronous replication state files on source server:
The source server creates the following tables to support synchronous replication:
```

Compatibility Notes:

To use this feature, the source and target servers and the Replication Agent must all include support for it. A new Replication Agent could work with old servers as long as it's not configured to use synchronous replication.

A field was added to the snapshot structure, so the system snapshot version has increased from 21 to 22. This field's value is displayed in *SNAPSHOT.FCS* for a text snapshot. The snapshot parse utility, **ctsnpr**, has been updated to handle this new *SNAPSHOT.FCS* format.

8.2 Client Failover Notifications

c-tree ISAM clients maintain a tight affinity to their connected server for rich context information. When failing over to a replica server, they need to re-establish this context information, which requires an entirely new connection to the failover instance. It is not always possible for a client application to know when its server has failed over. For example, when using a virtual IP address mapped to different server after failover, a client may hang as it is no longer a valid network client socket connection. A coordinated "notification" is needed informing a client application it must reconnect to continue processing data.

A new back-channel client listening thread is now available to provide this notification. A client can optionally enable this background thread before establishing its server connection. Once enabled, the client application can receive a specific notification when it is required to reconnect. This background thread listens on a configurable network secure UDP port. A specific failover notification message informs all listening clients on the network with a network broadcast message.

Three new client API calls make up this new client failover feature.

ctSetClientLibraryOption()

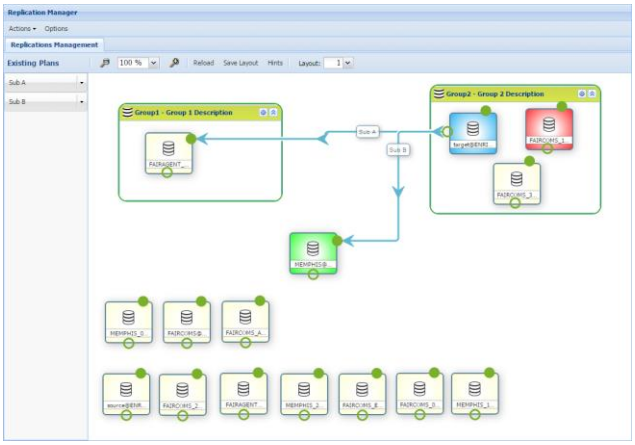
(https://docs.faircom.com/doc**/ctreeplus/ctsetclientlibraryoption.htm**) **ctGetFailOverState()**

(https://docs.faircom.com/doc**/ctreeplus/ctgetfailoverstate.htm**)

ctResetFailOverState() (https://docs.faircom.com/doc**/ctreeplus/ctresetfailoverstate.htm**)

9. Data Replication

Now available with FairCom DB V12, FairCom RTG V3, and FairCom Edge V3, the next phase of FairCom replication brings new capabilities to effortlessly tackle the most demanding of replication requirements:



- **Dynamic Data Sync** - The initial synchronizing of data between a source and target is no longer a manual process. Data synchronization is now under Replication Manager control and completely automated when deploying replication plans between servers.
- **Replicated File Operations** - File create, delete and rename operations are now under replication control.
- **Integrated Replication** - The new, advanced version of the Replication Agent is now a plug-in that runs inside the FairCom Database Process.
- **Synchronous Replication** - For mission-critical environments, this ensures data is replicated before returning a commit to the application, eliminating data loss and out-of-sync data.
- **Parallel Replication** - Replication is now multi-threaded for greater speed. You control the level of parallelism to meet your tolerance for latency, and to match the capabilities of your CPU. Parallel replication honors the order of dependent transactions to ensure full ACID compliance.
- **Replication Manager** - This browser-based interface does all the work to set up replication. It backs up source tables, creates tables on the target server, restores tables to the target server, creates the ongoing replication task, starts the replication task, and catches up all activity when replication is paused and restarted.

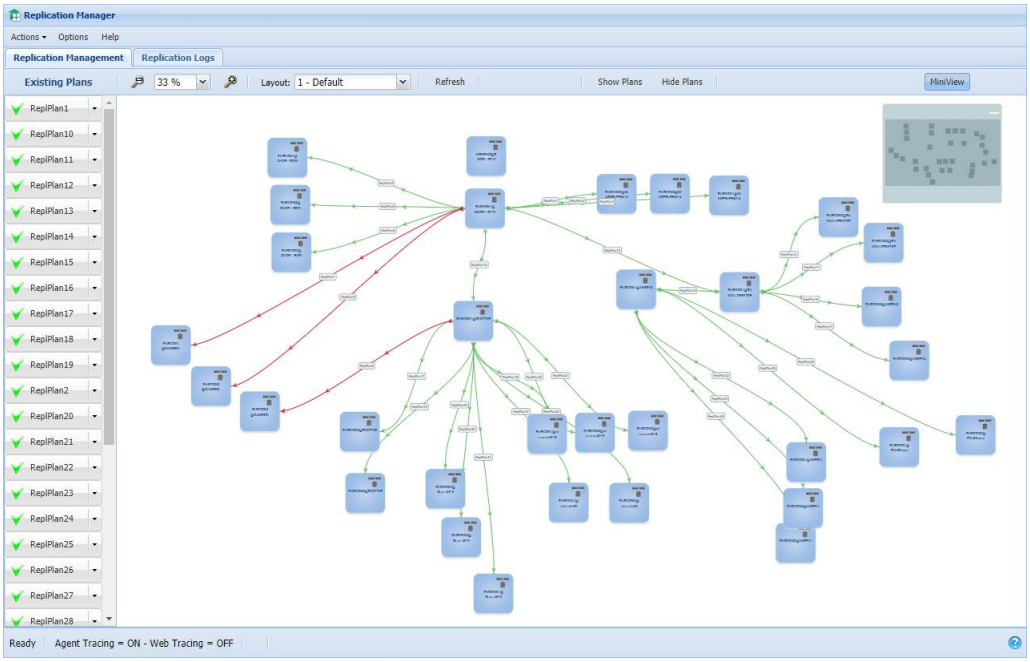
FairCom Replication has made such great strides in this release, and here are books where you can learn more details. Details on this advanced replication management facility are available here:

- *Replication Concepts* (https://docs.faircom.com/doc/*cluster-for-scalability-availability/*)
- *Replication Manager User Guide* (https://docs.faircom.com/doc/*replication-manager/*)
- *Replication JSON RPC API Developer's Guide* (https://docs.faircom.com/doc/*java-json-rpc-replication-api/*)
- *Replication Extensions* (https://docs.faircom.com/doc/*replication_extension/*)

9.1 Advanced Replication Keeps Your Data World in Sync

FairCom's replication solution is more powerful and flexible than ever. A new architecture provides unequaled performance and manageability. The FairCom replication solution uses a distributed architecture based on centralized metadata managed by the FairCom Replication Manager.

The Replication Manager allows you to access and control your distributed c-tree ecosystem from a central location. It provides a convenient, graphical interface for managing systems of any level of complexity at an individual node level.



For unattended deployment and automated administration, all Replication Manager functionality is available as an API. Two APIs are supported: A C++ API and a JSON RPC (Remote Procedure Call) that works over HTTP. By choosing one of these APIs, Database Administrators can build full-featured replication models for their exact needs.

This new release also replicates DDL information (file creates, deletes, and renames).

This release adds support for synchronous replication for immediately consistent data availability between FairCom Servers. The original and successful asynchronous model remains available for environments that benefit from the flexibility and increased performance it provides.

- Failover environments where the secondary (target) server must be always in sync with the primary (source) server.
- Environments where no committed transaction can be lost while moving from the primary to the secondary server.
- Environments where no delays are acceptable while moving to the secondary server.
- High Availability / Failover for every user scenario:
  Automatic OS Failover - To prevent data loss during failover by leveraging OS clustering.

Database-Level Automatic Failover *(Beta)* - FairCom can automatically notify the application of an outage so it can connect to a secondary server without relying on OS clustering.

Parallel Asynchronous Replication - Replication is now multi-threaded, making it much faster. And you can change replication plans without stopping the entire database engine.

Read Scalability - Provide a duplicate copy of your data in a geographically distant region with minimal performance impact.

Massive Read Scalability - Provide multiple duplicate copies of your data without impacting performance.

Disaster Recovery - Save an up-to-date backup copy of your data on a remote server.

Global Read Scalability & Disaster Recovery - Replicate the database across many servers, horizontally scaling it to large numbers of read-only users. Replicas act as backups for disaster recovery.

Global Shared Data - Selectively replicate tables and sharded data between servers to provide data across globally distributed applications.

Microservice Shared Data - Avoid slow joins across multiple microservices by providing each with its own data, isolated from other microservices.

Synchronous Parallel Replication *(Beta)* - The foundation of high availability, Synchronous Parallel Replication ensures all data committed to the primary server is also committed to the secondary, eliminating data loss.

Complete details on FairCom's replication advances are available in our new *Replication Concepts* book.

9.2 Dynamically Set Replication Node ID (REPL_NODEID) at Runtime

A replication node ID value (REPL_NODEID) associates a Replication Agent to a specific source or target server. Having this required association prevents Replication Agents from arbitrarily connecting to inappropriate servers. Setting a REPL_NODEID required modifying the server configuration file, *ctsrvr.cfg*, and restarting a server, which is challenging in round-the-clock production environments.

c-tree Server now supports setting a `REPL_NODEID` value dynamically at runtime if it has not yet been set. This keyword sets the node ID of a local c-tree Server for use in replication.

To set the value, use the **ctadmn** utility's option to change a server configuration option value using option 10 or call the **ctSETCFG()** function as shown here:

```
NINT rc;
rc = ctSETCFG(setcfgCONFIG_OPTION, value);
```

where value is the `REPL_NODEID` value to set, specified as a NULL-terminated string in IPv4 format (n.n.n.n.), for example "`10.0.0.1`". (ID values are arbitrary and do not need to match the IP address of the server.)

Note that the value is only changed in memory and not persisted to disk. Be sure to update the configuration file such that the new value is persisted and enabled on the next server startup.

10. Enhanced Security

Nefarious actors grow in number every day challenging every application. FairCom DB continues to extend security options for flexible data protection options. Enhanced options for both encryption of data at rest and data in flight are included in V12. Additional authentication and authorization controls are also included. FairCom strongly encourages application providers to take advantage of all options available, first and foremost, changing your default Administrator account password.

10.1 OpenSSL Now Provides Default Faster AES Encryption

FairCom now supports an updated AES algorithm based on the OpenSSL standard by default. Previous algorithm implementation can be restored by specifying in *ctsrvr.cfg* the keyword `OPENSSL_ENCRYPTION NO`.

This change is expected to provide security and, with internal testing, we have seen an up to 20% performance improvements.

This modification changes file passwords encrypted on the client-side for secure transmission to use OpenSSL.

10.2 Master Key Storage Integration with Amazon AWS Secrets Manager

(This technology is not included in the default package. It is available upon request.)

Master Encryption Key management is challenging. Keys must be remembered or securely retained, or data is permanently lost. To aid secure master key storage, FairCom Server can now integrate with an AWS Secrets Manager® service. This support requires an AWS account and a configured Secrets Manger service, which is not included nor maintained by FairCom.

Please contact FairCom with your specific Advanced Encryption Key management request for Amazon AWS support.

Support for Using AWS Secrets Manager as External Encryption key Store

FairCom Server now supports retrieving its master encryption key from AWS Secrets Manager.® This option is an alternative to FairCom Server's `MASTER_KEY_FILE` option. It has the advantage that the key is not stored locally. To logon to the AWS Secrets Manager, FairCom Server displays a dialog box that prompts the user to enter the AWS Secrets Manager credentials.

To configure FairCom Server to use AWS Secrets Manager, add the following option to *ctsrvr.cfg*:

```
SUBSYSTEM EXTERNAL_KEY_STORE AWS {
 KEY_ID <key_id>
 REGION <region>
 TIMEOUT <timeout>
}
```

<key_id> is the key ID that you assign to the encryption key when you store it in AWS Secrets Manager.

<region> is the AWS region where the AWS Secrets Manager stores your key.

<timeout> is the maximum amount of time, in seconds, to wait for the user to enter the AWS Secrets Manager credentials when FairCom Server starts. The default is 30 seconds. If the timeout period passes before the user enters the credentials, FairCom Server logs the following messages to *CTSTATUS.FCS* and shuts down:

```
 - User# 00001  Failed to retrieve AWS credentials: CTAWS(324): abandoning wait for AWS credential prompt to complete after <seconds> second(s) due to timeout
 - User# 00001  The master password must be entered in order to start this server.  The server will now shut down.
```

How to use AWS Secrets Manager to store the master encryption key for c-tree Server:

1. Login to AWS Secrets Manager.
2. Select "**Store a new secret**."
3. Select secret type of "**Other type of secrets**."
4. Enter the key and its value. For the key, use the name **ctreeServerMasterEncryptionKey**. For the value, enter the encryption master key.
5. Select encryption key of "**DefaultEncryptionKey**" and click **Next**.
6. Give the secret a name. This name is the value that you will specify for the KEY_ID value in the *ctsrvr.cfg* file.
7. Optionally add a description. Click **Next**.
8. Disable automatic rotation and click **Next**.
9. Click **Store**.

In standalone mode (**ctrdmp** utility for example), the configuration options are set by setting these environment variables to the desired values:

```
CTAWS_KEY_ID
CTAWS_REGION
CTAWS_TIMEOUT
```

If FairCom Server is configured to use AWS Secrets Manager and the DLL cannot be loaded, an error is logged to *CTSTATUS.FCS*:

```
- User# 00001  Could not load AWS support: CTDLL_LOAD: Failed to load module ctaws.dll: The specified module could not be found.: 981
```

FairCom Server supports changing the master encryption key when it is stored in AWS Secrets Manager.

DLL for FairCom Server to Access AWS Secrets Manager

A new DLL, **ctaws.dll**, exports a set of API functions that FairCom Server uses to access AWS Secrets Manager on Windows. FairCom Server dynamically loads this DLL when it is configured to use AWS Secrets Manager to store its master encryption key.

This DLL requires the Amazon provided AWS C++ SDK DLLs *aws-cpp-sdk-core.dll* and *aws-cpp-sdk-secretsmanager.dll*, which contain the AWS core and Secrets Manager functions respectively.

Product Installation Requirements:

To use AWS Secrets Manager as FairCom Server's encryption key store, the DLLs *ctaws.dll*, *aws-cpp-sdk-core.dll*, and *aws-cpp-sdk-secretsmanager.dll* must be installed in the same directory as the FairCom Server binary.

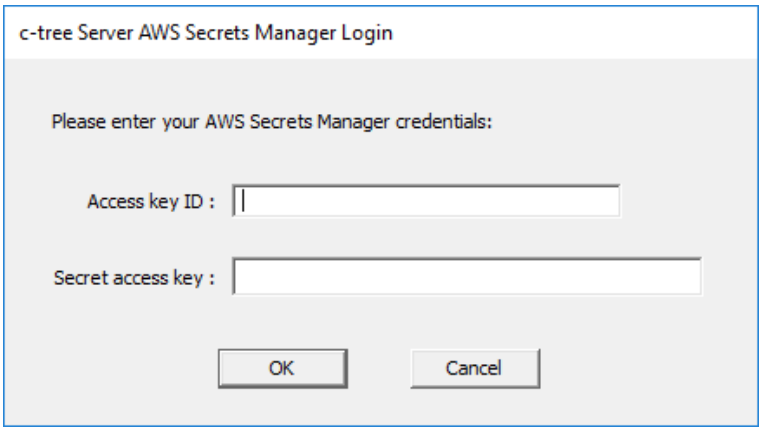
Note: When using AWS key store support on servers that are licensed for multiple remote desktop sessions, the password prompt may not be presented unless the database process is run as a normal process (not a Windows service). Because this might involve file permissions changes, it is recommended to install the database in this non-service configuration on such systems.

Visual Prompt Utility for AWS Credentials

To read FairCom Server's master encryption key from AWS Secrets Manager, we prompt the user to enter the AWS credentials using a graphical interface on Windows.

The **ctawssmp** utility is a process that displays a dialog prompting the user to enter the AWS Secrets Manager login credentials. FairCom Server launches this process when it starts up if it is configured to use AWS Secrets Manager as its encryption key store.

This Windows application displays the following message which prompts the user to enter the AWS Secrets Manager credentials:



Access Key ID and **Secret Access Key** are randomly-generated values created by an AWS account administrator. They require the proper access permissions to the AWS Secrets Manager to be able to read the secret referenced by the `KEY_ID`. For details about these values, see the Amazon document: https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

Product Installation Requirements:

To use AWS Secrets Manager as FairCom Server's encryption key store, **ctawssmp.exe** must be installed in the same directory as the FairCom Server binary.

10.3 Encrypted Data Master Key Library

It is now possible to implement custom solutions for retrieving the advanced encryption master key from an arbitrary library. This feature eases the way the developers can customize the master key prompt.

The new *ctsrvr.cfg* configuration keyword, `MASTER_KEY_LIB`, takes a string defining the complete library name to load, for example:

```
MASTER_KEY_LIB maskeylib.dll
```

or

```
MASTER_KEY_LIB libmaskey.so
```

The master key library must link with the OpenSSL libraries that are used to secure the master key exchange and implement the following functions:

- `int ctGetSecretVersion(void)` - returns the version of the master key library SDK used to implement it.
- `int ctGetSecret(ctGetSecretParams_t * GetSecretParams)` - returns the master key encrypted by calling **ctSecureMasterKey** as a member of the *ctGetSecretParams_t* structure.

Both functions are called by the server code in *ctcryp.c*. If the version does not match or **ctGetSecret** returns something different than 0, the master key will not be loaded and the server will be shut down.

To correctly return the encrypted master key, the following must be called to encrypt the master key before returning 0 in **ctGetSecret**:

```
int ctSecureMasterKey(ctSecureMasterKeyParams_t *SecureMasterKeyParams)
```

10.4 Ability to Validate against Advanced Encryption Master Password

FairCom Server now supports the ability to check if a client-provided master encryption password matches FairCom Server's current master encryption password. The use case for this feature is an elevated level of access beyond ADMIN authentication. Knowing the master encryption password implies the calling user has elevated privileges.

The server is assumed already started with a valid master key. A new **SECURITY()** (https://docs.faircom.com/doc**/ctreeplus/security.htm**) API mode makes this check against the current master key.

10.5 Automatically Enforce Password Strength

FairCom Server supports setting the following requirements for user account passwords:

- The minimum length of a password, which can be any value up to the maximum, 64.
- The minimum number of required character classes in a password, where the recognized character classes are: lowercase characters, uppercase characters, numbers, and symbols.
- The password expiration in days.

These options can be set server-wide by adding the following options to *ctsrvr.cfg*:

```
SUBSYSTEM SECURITY PASSWORD {
 MINIMUM_LENGTH   length
 REQUIRED_CLASSES classes
 EXPIRE_IN_DAYS   days
}
```

For example, to require passwords to be at least 8 characters, to require having at least 3 of the 4 character classes, and to expire passwords after 180 days, add the following to *ctsrvr.cfg*:

```
SUBSYSTEM SECURITY PASSWORD {
 MINIMUM_LENGTH   8
 REQUIRED_CLASSES 3
 EXPIRE_IN_DAYS   180
}
```

This subsystem can be specified in a server settings file to prevent these settings from being changed in the configuration file or at runtime.

If the minimum length and required classes settings are changed in the configuration file or at runtime, they do not affect existing passwords because only a hash of the password is stored. The time to expiration takes effect immediately even for existing user accounts.

When changing subsystem options at runtime (for example, using **ctadmn**), either the entire subsystem or particular subsystem options might be blocked by having been specified in a settings file. In both cases, the attempt to change the options fails with error **SETO_ERR** (804), even if some of the subsystem options are not blocked. If **SETO_ERR** occurs when specifying

multiple options at runtime, check *CTSTATUS.FCS* for a message indicating which options are blocked and try again with just those options that are not blocked.

10.6 SYSLOG Recording of SQL User Logon and Logoff Events

SYSLOG events have been added for SQL user logon and logoff events, as well as failed logon attempts. These are included in the *USER_INFO* option with other system logon/loggof events.

Add `SYSLOG USER_INFO` to your *ctsrvr.cfg* configuration to enable logging of these events to the SYSLOG event logs.

10.7 Function to Return User Account and Password Expiration Times

The c-tree client library function **ctGetAccountExpirationTime()** returns the user account and user password expiration times that are set for the specified user account. An ADMIN group user account can call this function for any user account. A user account that is not a member of the ADMIN group can only call this function for its own account, otherwise the function returns error **589**. If the specified user account does not exist, this function returns error **101**.

ctGetAccountExpirationTime()

(https://docs.faircom.com/doc**/ctreeplus/ctgetaccountexpirationtime.htm**)

10.8 Read-Only Server - Perfect for Reporting and Several HA (High Availability) and DR (Disaster Recovery) Scenarios

This option is available only if licensed. This configuration option sets the server in read-only mode. By default, this option is off. To enable this option, add `READONLY_SERVER YES` to *ctsrvr.cfg*.

This option can also be enabled or disabled at run-time by calling the **ctSETCFG()** function or using the **ctadmn** utility: Select menu option 10 and then menu option 10 (Change the specified configuration option) again to change a server configuration option value.

Runtime changes to this setting are persisted as documented in *Dynamic Configuration Changes Now Persisted* in *New ctsrvr.cfg Location and Default Additions (page 114).*

10.9 Advanced SSL Certificate Options

The `ssl_certificate` keyword in the *config/cthttpd.json* web server plug-in configuration supports the *fccert.pem*, which is a self-signed certificate we supplied. To use your own certificate, use the following keywords to *config/cthttpd.json*:

- *ssl_key: SSL private key* - This can be embedded in the same file provided the in *ssl_certificate*. For example, our default *fccert.pem* certificate file has both the certificate and the private key, so, *ssl_key* is not required.
- *ssl_ca: SSL Certificate Authority* - External authority that issues and validates the certificate.
- *ssl_cipher_suites* - Colon-delimited list of SSL cipher suites.

10.10 OpenSSL Support is Now Extended to Linux Platforms

OpenSSL libraries are statically linked to the server binary requiring two shared libraries:

- *libcrypto.so.1.0.0*
- *libssl.so.1.0.0*

Current TLS support is OpenSSL 1.0.2

Support for TLS / SSL (1.2) is available for Windows, Linux, and Mac OS X. Others operating systems considered upon request.

10.11 Perform LDAP_GROUP_CHECK in Context of LDAP Application ID if Specified

The check for group membership, configured by the `LDAP_GROUP_CHECK` option, was done in the context of the user account that was logging on. However, the user account might not have permission to query its LDAP groups.

The logic has been enhanced so that, if an LDAP application is specified (by specifying the `LDAP_APPLICATION_ID` option in the `SUBSYSTEM USER_AUTH LDAP` block in *ctsrvr.cfg*), it now performs the `LDAP_GROUP_CHECK` in the context of the LDAP application ID. This is consistent with what is done for the `LDAP_ISAM_ALLOWED_GROUP` and `LDAP_SQL_ALLOWED_GROUP` options.

When `LDAP_APPLICATION_ID` is specified, you MUST also use `LDAP_KEY_STORE` to specify an application password, otherwise the application authentication will fail.

Note: When `LDAP_APPLICATION_ID` is not specified the logic behaves as before, using the current user ID for lookup.

10.12 LDAP Authentication Diagnostic Logging

LDAP diagnostic logging is now available. LDAP diagnostic log messages are written to *CTSTATUS.FCS* and start with "`LDAP_DIAG:`"

Recommendation: It is best to use this feature only when resolving connection issues and then turn it off for production use. This practice minimizes the increase in information this feature writes to the log.

Specify `DIAGNOSTICS LDAP` in *ctsrvr.cfg* to enable the diagnostic logging at server startup. This logging can be enabled at runtime using **ctadmn**:

```
10. Change Server Settings
9. Change a DIAGNOSTICS option
Enter the DIAGNOSTICS option to enable or disable >> LDAP

(or ~LDAP to turn it off)
```

The annotated example below shows LDAP diagnostic logging messages that are written when a user connects and its LDAP groups are checked and updated in *FAIRCOM.FCS* (when the `LDAP_GROUP_CHECK` configuration option is used):

The `LDAP_GROUP_CHECK` setting:

```
 - User# 00020 LDAP_DIAG: chkldapusr: LDAP_GROUP_CHECK=[(objectclass=groupOfNames)]
 - User# 00020 LDAP_DIAG: chkldapusr:
pgroupflt=[(&(objectclass=groupOfNames)(member=cn=jeff,ou=people,dc=faircom,dc=com))]
```

The group base and filter that are sent to the LDAP server:

```
 - User# 00020 LDAP_DIAG: getldapusergroups: grp=[ou=groups,dc=faircom,dc=com],
flt=[(&(objectclass=groupOfNames)(member=cn=jeff,ou=people,dc=faircom,dc=com))]
 - User# 00020 LDAP_DIAG: getldapusergroups: ngroups=[5]
```

The groups read from the LDAP server:

```
 - User# 00020 LDAP_DIAG: getldapusergroups: group[0]: dn=[cn=dev,ou=groups,dc=faircom,dc=com]
 - User# 00020 LDAP_DIAG: getldapusergroups: dp=[dev]
 - User# 00020 LDAP_DIAG: getldapusergroups: group[1]: dn=[cn=support,ou=groups,dc=faircom,dc=com]
 - User# 00020 LDAP_DIAG: getldapusergroups: dp=[support]
 - User# 00020 LDAP_DIAG: getldapusergroups: group[2]:
dn=[cn=qalongerthanoursupportedmaximumgroupname,ou=groups,dc=faircom,dc=com]
 - User# 00020 LDAP_DIAG: getldapusergroups: dp=[qalongerthanoursupportedmaximumg]
 - User# 00020 LDAP_DIAG: getldapusergroups: group[3]: dn=[cn=ctreeisamusers,ou=groups,dc=faircom,dc=com]
 - User# 00020 LDAP_DIAG: getldapusergroups: dp=[ctreeisamusers]
 - User# 00020 LDAP_DIAG: getldapusergroups: group[4]: dn=[cn=ctreesqlusers,ou=groups,dc=faircom,dc=com]
 - User# 00020 LDAP_DIAG: getldapusergroups: dp=[ctreesqlusers]
```

The groups returned to the calling function:

```
 - User# 00020 LDAP_DIAG: chkldapusr: numldapgroups=5
 - User# 00020 LDAP_DIAG: chkldapusr: ldapgroup[0]=[dev]
 - User# 00020 LDAP_DIAG: chkldapusr: ldapgroup[1]=[support]
 - User# 00020 LDAP_DIAG: chkldapusr: ldapgroup[2]=[qalongerthanoursupportedmaximumgctreeisamusers]
 - User# 00020 LDAP_DIAG: chkldapusr: ldapgroup[3]=[ctreeisamusers]
 - User# 00020 LDAP_DIAG: chkldapusr: ldapgroup[4]=[ctreesqlusers]
```

The updating of groups in *FAIRCOM.FCS*:

```
 - User# 00020 LDAP_DIAG: updatectreeusergroups: ctreegroups=0, ldapgroups=5
 - User# 00020 LDAP_DIAG: updatectreeusergroups: deleted all c-tree groups for user [JEFF]
 - User# 00020 LDAP_DIAG: updatectreeusergroups: added c-tree group [CTREEISAMUSERS]
 - User# 00020 LDAP_DIAG: updatectreeusergroups: added c-tree group [CTREESQLUSERS]
 - User# 00020 LDAP_DIAG: updatectreeusergroups: added c-tree group [DEV]
 - User# 00020 LDAP_DIAG: updatectreeusergroups: added c-tree group [QALONGERTHANOURSUPPORTEDMAXIMUM]
 - User# 00020 LDAP_DIAG: updatectreeusergroups: added c-tree group [SUPPORT]
```

10.13 .NET FairCom.Isam Updated for Security Features

The .NET FairCom.Isam DLL has been updated to support new security features, such as password expiration. A password expiration member (*usr_xpwdexp*) has been added to the USRINFO structure. User-related functions have also been added.

10.14 V12 Changes

With V12, FairCom has strengthened it's license file checks. Therefore, before rolling out a V12 production license (ctsrvr*.lic file), we recommend thorough testing on your production machine to

ensure your license files are properly sized for your production hardware. Details on CPU counting and licensing can be found here (https://docs.faircom.com/doc/ctserver/CountingCPUs&Threads.htm).

FAIRCOM.FCS V9 and older must be recreated:

FairCom DB V12 and FairCom RTG V3 have deprecated an older algorithm that prevents usage of *FAIRCOM.FCS* files from FairCom DB V9 and older Server lines (and any customers using the *prev10logon* switch within their V10 and newer *ctsrvr*.lic* files). The solution is to recreate the *FAIRCOM.FCS* file by not moving this file forward.

Note: Existing user IDs and passwords will need to be recreated with V12 and V3.

See **Adjusting PAGE_SIZE (https://docs.faircom.com/doc/FairCom-Installation/AdjustingPAGE_SIZE.htm).**

11. Develop Easier

Many V12 new features and enhancements required new or enhanced API calls. The list is extensive!

New APIs

- JSON (page 79)
- REST (page 89)
- MQTT (page 94)

Updated functionality

- c-treeDB Functions (page 108)
- ISAM Functions (page 110)
- Low-Level Functions (page 111)
- System Functions (page 111)
- SQL Functions (page 113)
- Direct SQL Functions (page 113)

11.1 100+ New and Enhanced Development Features

Check out the full listing of new and updated c-tree API Functions (page 108).

11.2 New and Enhanced APIs and Drivers

JSON

Query JSON from SQL

JSON support in SQL provides the ability to read/write JSON field types mapped into VARCHAR (if the size is less than 64K) or LONGVARCHAR.

Use JSON in Your c-treeDB Applications

New c-treeDB Functions for JSON

- **ctdbGetFieldAsJSON()** (https://docs.faircom.com/doc/**ctreedb/ctdbgetfieldasjson.htm**)
- **ctdbSetFieldAsJSON()** (https://docs.faircom.com/doc/**ctreedb/ctdbsetfieldasjson.htm**)
- **ctdbAddJSONSegment()** (https://docs.faircom.com/doc/**ctreedb/ctdbaddjsonsegment.htm**)
- **ctdbSetJSONSegmentField()** (https://docs.faircom.com/doc/**ctreedb/ctdbsetjsonsegmentfield.htm**)

New c-treeDB Error Code

CTDBRET_FIXJSON = 4168 /* JSON field cannot reside in fixed record portion */

New c-treeDB Data Type

```
typedef pTEXT   CTJSON, ctMEM *pCTJSON;
```

Using JSON Data Types in Your ISAM Applications

JSON Data Type

To create a table that contains a CT_JSON field, specify CT_JSON as the field type in the DODA. When you add a record that contains a CT_JSON field value, store it in CT_4STRING format, with the first four bytes of the field containing the length of the JSON value, followed by the JSON value.

JSON Indexing

To create an index on a CT_JSON field, set the index attributes in the IIDX and ISEG structures that you pass to **CREIFIL()**. Before calling **CREIFIL()**, call the **PUTKSEGDEF()** (https://docs.faircom.com/doc**/ctreeplus/putxtdkeysegmentdef.htm**) function to set the CT_JSON index attributes that will be used for the CT_JSON index in the **CREIFIL()** call. The CT_JSON index attributes include the key segment length, the c-tree data type of the key value, and the JSON key name that is
being indexed.

The **PUTKSEGDEF()** (https://docs.faircom.com/doc**/ctreeplus/putxtdkeysegmentdef.htm**) function can be used to specify CT_JSON index attributes for a call to **CREIFIL()**:

1. Declare and initialize an array of pointers to ctKSEGDEF structures.
2. Pass this array of pointers to **PUTKSEGDEF(***filno,segno,pkdef***)** by setting *filno* to ctKSEGcreidx, setting *segno* to the number of pointers in the array, and setting *pkdef* to the address of the array. Note that the array you pass must contain a pointer for every key segment that precedes the CT_JSON key segment. A pointer can be NULL if no extended index information is to be supplied for that key segment.

Example

In this example, the supplied IFIL definition (*pifil*) has two indexes of one key segment each. The example code sets the first extended segment info pointer to NULL and the second to a ctKSEGDEF structure initialized with the CT_JSON index attributes.

```
ISEG myseg[] = {
 { 8,  8, SRLSEG},
 { 3,  4, SCHSEG}
};
IIDX myidx[] = {
 {8, 0,  NO, NO, 0, 1, myseg, "$ROWID$"},
 {4, 0,  NO, NO, 0, 1, myseg+1, "doc_id"}
};
IFIL  mydat = {
 "qajson",
 -1,
 20,
 0,
 ctFIXED | ctTRNLOG,
 3,
 0,
 ctTRNLOG,
 myidx
};
ctKSEGDEF sd;
pctKSEGDEF pksegs[2];

/* No extended segment info for first index. */
pksegs[0] = NULL;

/* Set the extended segment info for the second segment. */
memset(&sd,0,sizeof(sd));
sd.kseg_type = ctKSEG_TYPE_JSON;
/* use source type from schema */
sd.kseg_styp = ctKSEG_STYP_PROVIDED;
/* set key segment length */
sd.kseg_ssiz = 4;
/* set c-tree data type of key value */
sd.kseg_comp = CT_INT4;
/* set the json key name that is being indexed */
strcpy(sd.kseg_desc,"id");
pksegs[1] = sd;

rc = CREIFIL(pifil);
```

The name of the JSON key to index is stored in the *kseg_desc* field of the ctKSEGDEF structure. However, this field is limited to ctKSEGDLEN (32) bytes in the ctKSEGDEF structure definition. To overcome this limit, the ctKSEGcreidx usage of **PUTKSEGDEF()** (https://docs.faircom.com/doc**/ctreeplus/putxtdkeysegmentdef.htm**) accepts an array of pointers to ctKSEGDEF
structures instead of an array of ctKSEGDEF structures so that an application can allocate a

buffer to hold the CT_JSON index attributes that is larger than the ctKSEGDEF structure when the JSON key name exceeds the length of the field as defined in the ctKSEGDEF structure. As long as *kseg_desc* is NULL-terminated, the larger *kseg_desc* value can be passed to **PUTKSEGDEF()** in this manner.

The ctKSEGcreidx information that is passed to **PUTKSEGDEF()** is stored in the c-tree library until the next call to an ISAM level index create function: **CREIFIL()**, **CREIFILX()**, **CREIFILX8()**, **PRMIIDX()**, **PRMIIDX8()**, **TMPIIDX()**, **TMPIIDXX()**, **TMPIIDXX8()**. The recommended usage is to call **PUTKSEGDEF()** with *filno* of ctKSEGcreidx immediately before the index create call.

For an extended key segment type (*kseg_type*) of ctKSEG_TYPE_JSON, a call to **GETKSEGDEF()** is able to return to the caller a larger amount of ctKSEGDEF data than the size of the ctKSEGDEF structure. A call to **GETKSEGDEF()** for a *kseg_type* of ctKSEG_TYPE_JSON should set *kseg_ssiz* to the size of the supplied buffer. If **GETKSEGDEF()** finds that the data is larger than the supplied buffer size, it returns **-VBSZ_ERR** and returns the required length in the *kseg_rsv1* field. For example:

```
ctKSEGDEF sd;
NINT rc,bufsiz;

sd.kseg_type = ctKSEG_TYPE_JSON;
sd.kseg_ssiz = bufsiz;
if ((rc = GETKSEGDEF(keyno,i,&sd)) < 0) {
 if (rc == -VBSZ_ERR) {
  /* Buffer too small. sd.kseg_rsv1 holds the required size. */
 }
}
```

This new behavior of **PUTKSEGDEF()** and **GETKSEGDEF()** is intended to be used to send and receive attributes for other extended key segment types in the future.

JSON Error Code

Error code **1122**:

```
JSON_KTFM_ERR: CT_JSON key transformation failed. Check sysiocod for details. See CTJSON_* definitions.
```

sysiocod values that may be returned in case of error **1122**:

```
#define CTJSON_NOT_JSON  -1038 /* Not a valid JSON object */
#define CTJSON_NOT_OBJECT  -1039 /* Valid JSON, but not an object */
#define CTJSON_UNSUPPORTED_TYPE -1040 /* The specified key data type is not supported */
```

Limitations to JSON Key Segments

1. A CT_JSON key segment does not support the DSCSEG, ALTSEG, or ENDSEG key segment modes. Attempting to create an index that contains a CT_JSON key segment that includes any of these key segment modes fails with error **NSUP_ERR**.
2. CT_JSON key segments that use CT_STRING as the c-tree data type are always padded with null (0x0) bytes. We don't support setting a different key segment padding character on JSON key segments.
3. A CT_JSON key segment can specify any of the following c-tree data types: CT_BOOL, CT_CHAR, CT_CHARU, CT_INT2, CT_INT2U, CT_INT4, CT_INT4U, CT_INT8,

CT_DFLOAT, CT_STRING. Note that CT_INT8U is not supported, because JSON only supports a signed 8-byte integer data type.

4. Remember that the JSON field indexing behaves as follows: if the JSON data type of the value that is being indexed is not compatible with the underlying c-tree data type specified in the key segment definition, that value is treated as NULL and is not indexed. Some examples:
   - If the value is a string and the c-tree data type is an integer, the value is not indexed.
   - If the value is an integer and the c-tree data type is CT_INT2 but the value is out of range for a signed two-byte integer, the value is not indexed.

JSON Data Type Support

V11.8 and later added JSON as a new data type, called CT_JSON. It is physically stored in a table using the CT_4STRING format, which can support strings up to 2 GB in length.

```
{
  "id": 900,
  "customerId": 7,
  "orderDate": "2019-12-19",
  "orderedProducts": [
    {"id": 81},
    {"id": 82}
  ]
}
```

It is available in all APIs with varying levels of support: SQL, NAV, ISAM, and Low-Level.

The NAV API allows you to index specific JSON properties using FairCom's path language. This allows you to find records containing specific property values or ranges of property values. And it allows you to navigate records in sorted order using JSON property values.

In SQL and ISAM, you can read and write the string value stored in a JSON field but you cannot extract the values in JSON properties and treat them as columns. You cannot also use indexes created by the NAV API.

JSON Supported in REST API

The REST API supports a JSON field type. c-treeDB also supports the ability to index JSON field contents.

To index a JSON field, specify the JSON field name in the "name" attribute and the key, type and length of the attribute that needs to be indexed using the "key", "type" and "size" attributes in the index definition.

Example:

```
{
    "fields": [{
            "name": "custinfo",
            "key": "name",
            "type": "string",
            "size": 50
    }],
    "unique": false
}
```

When inputting a record with a JSON field, the JSON field value can be specified as a JSON type. This is the highly recommended method:

```
{
    "custid": "1000",
    "custinfo": {
            "name": "Bryan Williams"
    }
}

Although not recommended, when inputting a record with a JSON field, the JSON field value can be specified
as a string representing the JSON to insert:

"{\"custom\":\"1000\",\"custinfo\":{\"name\":\"Bryan Williams\" }"
```

Below is an example on how to query the JSON content once indexed:

```
{
  "find": {
    "operator": ">=",
    "fields": {
      "custinfo": {"name":"a"}
    }
  },
  "select": [
    "custid",
    "custinfo"
  ]
}
```

When creating a table with a JSON field, the field length specified is ignored and the JSON field content can be as large as 2GB.

JSON Types

Because JSON supports a very precise list of types, the following types are included in JSON support:

- *integer* - signed integer with size 1, 2, 4; any other size is considered as 8
- *unsigned* - unsigned integer with size as integer
- *string* - a string with the specified size
- *float* - same as double
- *double* - double precision floating point, size forced to 8
- *boolean* - Boolean value, size forced to 1

MQTT Persistence API Can Map Nested JSON Structures to Multiple Tables

Support has been implemented for persisting a JSON homogeneous array of simple values into multiple records in children tables.

The `"mapOfPropertiesToFields"` array supports the following list of new properties:

- `childTableName` - Similar to `"fieldName"`, but the property will be persisted in a child table.
- `childFieldType`, `childFieldWidth`, `childFieldScale` - Similar to `"fieldType"`, `"fieldWidth"` and `"fieldScale"` but it will define the type of the "value" field in the child table. For example:

```
{
  "operation": "CreatePersistenceTopic",
  ""persistenceTopic": "myTopic",
  "tableName": "ex1",

  "mapOfPropertiesToFields":
  [
    {
      "propertyPath": "myProp1",
      "fieldName": "myProp1",
      "fieldType": "VARCHAR"
    },
    {
      "propertyPath": "myProp2",
      "fieldName": "myProp2",
      "fieldType": "DOUBLE"
    },
    {
      "propertyPath": "myStringArray",
      "childTableName": "ex1_mystringarray",
      "childFieldType": "VARCHAR",
      "childFieldWidth":64
    },
    {
      "propertyPath": "myIntegerArray",
      "childTableName": "ex1_myintegerarray",
      "childFieldType": "INTEGER"
    }

  ]
}
```

This can be used for persisting the following payload example:

```
{
  "myProp1": "test1",
  "myProp2": 13.2,
  "myStringArray":
  [
    "string0",
    "string1",
    "string2"
  ],
  "myIntegerArray": [10, 11, 12]
}
```

If a persistence table has children tables, a `"primarykey"` field of type BigInt is automatically added to the main persistence table. It will be linked to the the `"foreignkey"` field in the children tables. Apart from that, the children table will have only one field, named "value",

containing the array's element information and a field named "position" with its position in the array.

For persistence tables with auto-purge, the children tables also have the `"ts"` field and they are also partitioned based on that field.

Multi-Column "fieldCount"

The elements of the `"mapOfPropertiesToFields"` array support a property named `"fieldCount"` (Default is 1). If this value is greater than 1, it will create multiple fields of the same data type with the following name: `<fieldName>N` (where N is 0, 1, ...). If the JSON property being persisted is an array, it will try to extract all the elements from the array and populate these fields. If the array has more elements than fields, the remaining fields are ignored. If the array has fewer elements than fields, the remaining fields are set to NULL.

Example of usage:

```
{
  "operation": "CreatePersistenceTopic",
  "persistenceTopic": "ford6",
  "tableName": "ford6",

  "mapOfPropertiesToFields":
  [
    {
      "propertyPath": "myProp1",
      "fieldName": "myProp1",
      "fieldType": "VARCHAR",
      "fieldWidth":64
    },
    {
      "propertyPath": "myProp2",
      "fieldName": "myProp2",
      "fieldType": "DOUBLE"
    },
    {
      "propertyPath": "myStringArray",
      "fieldName": "c",
      "fieldType": "VARCHAR",
      "fieldWidth":64,
      "fieldCount": 3
    },
    {
      "propertyPath": "myIntegerArray",
      "fieldName": "i",
      "fieldType": "INTEGER",
      "fieldCount": 3
    }

  ]
}
```

Can be used for persisting the following payload example:

```
{
  "myProp1": "test1",
  "myProp2": 13.2,
  "myStringArray":
  [
    "item0",
    "item1",
    "item2"
  ],
  "myIntegerArray": [10, 11, 12]
}
```

REST API

REST API Support for Auto-Purge

The Auto-Purge feature is available in the REST API. This feature requires a partitioned table, which is set up as shown below. You will define a rule that will determine the partition number (for example, you might use part of the datetime stamp to create a partition for every day or month). You will also specify the maximum number of active partitions. After this value has been exceeded, partitions will be deleted automatically.

To create a partitioned table, you need to create a "partition index" on an existing table. The JSON request syntax to create an index includes a *partition* property that allows specifying a partitioned file setting as follows:

```
partition   {
    description:
    Optional parameter. If set the index is a partition index and the table becomes a partitioned table
with the settings defined by this property


        rule*   string
        Expression that evaluates into an integer that will be used as partition number.


        numberofbits    integer
        Optional parameter. Set the number of bits used to store the raw partition number. By default, 16
bits of the 64-bit record offset are used to reference the raw partition number, allowing each partitioned
file to support up to 65535 member files over its lifetime. A value of 0 defaults to 16 bits, values less
than 4 bits default to 4 bits (maximum 15 member files), and 32 bits is the maximum value.


        maximumactive   integer
        Number of maximum active partitions. When this maximum is exceeded, c-tree's auto purging feature
takes care of purging.


    }
```

Below is an example of use:

```
{
    "unique": true,
    "fields": [
        {
            "name": "custnumb",
            "ascending": true
        }
    ],
    "partition": {
        "rule": "custnumb - 1",
        "numberofbits": 16,
        "maximumactive": 30
    }
}
```

The JSON returned when describing the table includes "remarks" information about the partitioned file setting. The "remarks" property is ignored when defining the table structure. Below is an example:

```
remarks {
    description:
        Read only optional property. Contains information about the table structure that are set
        thru other means than a table definition during creation

    partitioned {
        description:
        the table is a partitioned table

        partition_index string
        The name of the partition index

        rule string
        The partition rule

        numberofbits integer
        Optional parameter. Set the number of bits used to store the raw partition number.
        By default, 16 bits of the 64-bit record offset are used to reference the raw partition number,
        allowing each partitioned file to support up to 65535 member files over its lifetime.
        A value of 0 defaults to 16 bits, values less than 4 bits default to 4 bits (maximum 15 member
        files), and 32 bits is the maximum value.

        maximumactive integer
        number of maximum active partitions, when exceeding the maximum c-tree's auto purging feature
        takes care of purging
    }
}
```

Example:

```
{
  "fields": [
    {
      "name": "custnumb",
      "type": "INTEGER"
    },
    {
      "name": "custzipc",
      "type": "CHAR",
      "length": 9
    },
    {
      "name": "custstat",
      "type": "CHAR",
      "length": 2
    },
    {
      "name": "custrtng",
      "type": "CHAR",
      "length": 1
    },
    {
      "name": "custname",
      "type": "VARCHAR",
      "length": 47
    },
    {
      "name": "custaddr",
      "type": "VARCHAR",
      "length": 47
    },
    {
      "name": "custcity",
      "type": "VARCHAR",
      "length": 47
    }
  ],
  "remarks": {
    "partitioned": {
      "partition_index": "partidx2",
      "rule": "custnumb - 1",
      "numberofbits": 16,
      "maximumactive": 30
    }
  }
}
```

REST API autosystemtime for Automatic System Time

In V12 and later, support for Automatic System Time is available in the REST API to automatically set time/date/timestamp fields at record creation or update.

A new property, *autosystemtime*, can be set in the field information in the table object at field creation time to specify this support. Below is an example of a JSON request with auto system time support (see last entry):

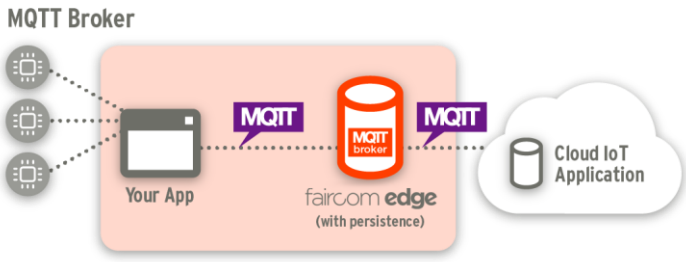
```
{
  "fields": [
    {
      "name": "custnumb",
      "type": "CHAR",
      "length": 4
    },
    {
      "name": "custzipc",
      "type": "CHAR",
      "length": 9
    },
    {
      "name": "custstat",
      "type": "CHAR",
      "length": 2
    },
    {
      "name": "custrtng",
      "type": "CHAR",
      "length": 1
    },
    {
      "name": "custname",
      "type": "VARCHAR",
      "length": 47
    },
    {
      "name": "custaddr",
      "type": "VARCHAR",
      "length": 47
    },
    {
      "name": "custcity",
      "type": "VARCHAR",
      "length": 47
    },
    {
      "name": "custtime",
      "type": "TIME",
      "autosystemtime": {
      "create": true
      }
    }
  ]
}
```

A new field property, *autotimestamp*, has been added to the table object to specify the support for *autosystemtime* assignment and the events (create and/or update) that trigger the assignment. Here is the OpenAPI specification:

```
Table
    description:
            Represents a single table.
    fields
            [
            minItems: 1
            {
                name*           string
                type*           string
                                Enum:
                                Array [ 21 ]
                length          integer
                scale           integer
                precision       integer
                autosystemtime  {
                                description:
                                        The field is automatically set with systemtime,
                                        create and update settings determine when the field
                                        is populated, at creation time or at update time.
                                create  boolean
                                update  boolean
                                }
        }]
}
```

MQTT

MQTT, or Message Queuing Telemetry Transport, is a lightweight protocol allowing small pieces of data to be transferred from machine to machine across networks. MQTT works well even on low bandwidth, high latency connections, which makes it ideal for the Internet of Things (IoT).



The FairCom MQTT Plug-in provides an MQTT Broker with the ability to persist MQTT messages in relational tables for easy integration with other systems.

MQTT communication consists of a publisher, a broker, and a subscriber:

- A *subscriber* connects to a broker and registers for a particular topic (a UTF-8 string used by the broker to filter messages for each connected client).
- A *publisher* connects to a broker and sends a message tagged with a particular topic.

- The *broker* receives the message and sends it to any subscribers who have subscribed to that topic.

With the MQTT Plug-in, the Broker is a generic broker for connecting c-tree as a persistent data store under an MQTT application. It can be placed on smaller devices and gateways typically found near sensors. It persists MQTT message contents on the edge. By locating c-tree on the edge, MQTT message contents can be persisted near the sensors where service is unaffected by the latency of the cloud.

Any JSON field contained in a specified topic is automatically persisted to a specified database table. This table is simultaneously available via a FairCom DB SQL server allowing ISAM and SQL connections to the persisted fields for immediately accessible data.

Data continues to flow through the Broker normally, going from publisher to subscriber while, at the same time, being stored in a database. Configure as many tables as you want each one collecting fields from different topics. This allows full SQL access to some or all fields of data moving through the MQTT Broker with very little change to standard MQTT setups.

Persistence Auto-Restart - When a MQTT Persistence is enabled, a file is placed in the *mqtt/active* directory with the topic name and persistence information. If the server is restarted with the persistence active, it loads this file during the initialization and restarts the persistence automatically. When Persistence is turned OFF, this file is deleted.

MQTT Using Auto Timestamp

When MQTT persistence has timestamp set, instead of populating the timestamp from the current time, it is populated using the *automatic timestamp* (page 97) feature. In this way, the "ts" field is now populated with the current Greenwich Mean Time (GMT).

MQTT - Option for Replication Enable/Disable

Support has been added to the MQTT Persistence API for a new optional property: `replicate`.

`replicate: true` automatically makes the table eligible for replication (default: true). Note that it increases the resources usage by the server.

If the MQTT persistence API receives `replicate: true`, or it is not sent any value for `replicate`, the *ctREPLICATEhdr* option is added to its header after the table creation. If this property is received as FALSE, the table is regular. Note that if the table has this property, the whole record content is logged in the transaction log.

Node.js

11.3 Introducing Node.js for NAV

Node.js for NAV is a direct navigational database interface for Node.js supporting all the features of the c-treeDB navigational interface ("NAV").

Examples are located in your FairCom DB *drivers* folder in *nodejs.nav*.

Example:

```
const ctdb = require('ctdbsdk');
const {deleteRecords} = require('./deleteRecords');
const {addRecords} = require('./addRecords');
var ref = require('ref-napi');

ctdb.Logon(hSession, "FAIRCOMS", "ADMIN", "ADMIN");

var hDatabase = ctdb.AllocDatabase(hSession);
ctdb.Connect(hDatabase, "ctreeSQL");

var hTable = ctdb.AllocTable(hDatabase);

try {
    console.log("Open table...");
    ctdb.OpenTable(hTable, "custmast", ctdb.CTOPEN_NORMAL);
} catch(e) {
    console.log("Open Failed...");
}
```

11.4 Node.js for SQL

Node.js for SQL allows FairCom DB to be used in a conventional Node.js environment.

Node.js 12 and later supported.

Examples are located in your FairCom DB *sdk* folder as *nodejs.sql*.

Example:

```
const ctsql = require('./ctsql');

ctsql.connect("6597@localhost:ctreeSQL", "admin", "ADMIN");
var result = ctsql.query("SELECT * FROM admin.syscolumns");

console.log(result.columnNames);
console.log(result.rows[0]);
console.log(result.rows[1]);

ctsql.end();
```

Specify character encoding at connection time

The Node.js interfaces need to properly handle strings with character encoding different than ANSI and UTF-8. An optional parameter for *ctsql.Connect* specifies the source encoding. If the parameter is present and correctly passed, each string read from the database is converted from the specified character encoding to the Node.JS default UTF-8 using the *iconv* module. (This implies a dependency on the *iconv* module.)

Automatic System Time Field Definition

As part of the support for automatic timestamps, the following new functions have been added to c-tree's API in V12 and later:

- **AddAutoSysTimeFields()** (https://docs.faircom.com/doc/**ctreeplus/addautosystimefields.htm**)
- **WhichAutoSysTimeFields()** (https://docs.faircom.com/doc/**ctreeplus/whichautosystimefields.htm**)
- **RemoveAutoSysTimeFields()** (https://docs.faircom.com/doc/**ctreeplus/removeautosystimefields.htm**)
- **UpdateAutoSysTimeFields()** (https://docs.faircom.com/doc/**ctreeplus/updateautosystimefields.htm**)

Python

SSL support parameter added to connect function

A new optional parameter has been added to the "connect" function as follows:

```
connect(**kw_args)
```

Constructor for creating a connection to the database.

Returns a Connection Object.

It takes parameters as keyword parameters for more intuitive use as follows:

- *user* - User name as string (optional default guest)
- *password* - Password as string (optional default None)
- *host* - Hostname (optional default localhost)
- *database* - Database name (optional default ctreeSQL)
- *port* - TCP/IP port number (optional default 6597)
- *ssl* - None: No SSL, (optional default None)
  `'BASIC'`: SSL without certificate checking,
  `'<certificate file name>'`: SSL with certificate checking, certificate file as specified,
  `''` (two single quotation marks): SSL with certificate checking, certificate in current working dir named *ctsrvr.pem*.

Python SQL enhancements

In FairCom DB Python, the following SQL enhancements have been implemented in the Cursor class:

- iterator
- *lastrowid* property

The following enhancements have been implemented in Connection class:

- support for constructor "port" parameter passed as an integer value
- support for **set_isolation** and **get_isolation** methods

Python support for Unicode server

The Python driver has been enhanced to detect the nature of the *ctsqlapi* library in use (Unicode vs non-Unicode) and consider this when calling into the library. The driver can now be used with a Unicode server as well as non-Unicode servers.

New SQLAlchemy support

FairCom is proud to announce support for SQLAlchemy in Python. SQLAlchemy makes it easy for you to use the FairCom Database Engine in your Python applications. This makes it trivial to switch to the FairCom Database Engine from another SQL database.

SQLAlchemy is an Object Relational Mapper for Python that makes it easy to map relational tables to Python objects. SQLAlchemy has earned a reputation for being the persistence mechanism of choice for use in Python environments.

This dialect has been tested with Python 2.7 and Python 3.5 & 3.6.

JDBC Conformance Updated to JDBC 4.3

JDBC conformance level 4.3 provides Java 9 and later compatibility.

PHP PDO SSL Security Added

A new parameter has been added to the connection string to indicate if SSL should be used:

- *ssl=BASIC* - SSL without certificate checking.
- *ssl=<certificate file name>* - SSL with certificate checking, certificate file as specified.

For example:

```
$ses = new PDO('ctsql:port=6597;host=localhost;dbname=ctreeSQL;ssl=BASIC', 'admin', 'ADMIN');
```

ODBC

Nodename + Process ID used to identify ODBC applications

The ODBC API does not have an interface to specify extra identifying information that our server can display to identify a specific application using the ODBC driver. By default, c-tree Server set the "nodename" to "SQL:<dbname>" when a new connection was made.

It is now possible to display a client process ID connected to the server. On the server side, when `SQL_OPTION PID_IN_NODEID` is specified in **ctsrvr.cfg**, the client process ID is appended to the default nodename as "SQL: PID <client pid>". If the client application does not pass the PID information, the original default nodemane is still used. The default node name can be overwritten at any time after connection by a client calling the **fc_set_nodename** (https://docs.faircom.com/doc**/sqlops/56378.htm**) built-in stored procedure.

This nodename string is displayed for all SQL client APIs and tools except for JDBC-based APIs/tools.

Log statements for DESCRIBE operation

The ODBC driver has been updated so that statements sent by the DESCRIBE operation are now being logged. The logging logic has been modified to always specify the operation the server is executing to make it easier to reconcile statements logged more than once.

11.5 Millisecond Timestamps

High-Resolution Timestamp Support Added

Two c-treeDB API functions have been added for high-resolution timestamp generation:

- **ctdbCurrentTimestamp()** (https://docs.faircom.com/doc**/ctreedb/ctdbcurrenttimestamp.htm**)
- **ctdbCurrentDateTimeUTC()** (https://docs.faircom.com/doc**/ctreedb/ctdbcurrentdatetimeutc.htm**)

Millisecond Time Format Added as hh:mm:ss:ttt

A new time format has been added to the CTTIME_TYPE enum:

```
CTTIME_HHMST - Time is hh:mm:ss.ttt (24 hour)
```

This format forces the hour to two digits. This is similar to CTTIME_HHMS, however this format returns the milliseconds component. Related functions have been updated to support this new format:

- **ctdbTimeMsecToString()**
- **ctdbStringToTime()** (https://docs.faircom.com/doc**/ctreedb/ctdbstringtotime.htm**)
- **ctdbStringToDateTime()**
- **ctdbDateTimeToString()**

11.6 Store UTF-8 in String Types

Using c-treeDB with UTF-8 encoded data works as expected with standard string fields. However, it was not possible to identify those string fields containing UTF-8 data to SQL on import. Proper international character support dictates that each string field should have charset and collation attributes associated with it.

c-tree has different field types to distinguish between UTF-16 strings (CT_*UNICODE) and "generic strings", however, it does not know about the character set and encoding used for generic strings. c-tree at its core is not concerned with this issue as the application itself normally directly handles these strings as bytes. However, high-level languages such as JAVA and SQL require defined string encoding for correct handling.

Functions have been added to c-treeDB for setting and identifying field level string character encoding.

ctdbSetFieldStringEncoding

Set the field string encoding.

```
CTDBRET ctdbDECL ctdbSetFieldStringEncoding( CTHANDLE Handle, pTEXT encoding )
```

Parameters:

- *Handle* [IN] - A c-treeDB associated field handle
- *encoding* [IN] - A string describing the encoding used for the field content. c-treeDB does not enforce any check on the value passed in, however, JTDB, JDBC, ADO.Net providers and SQL expect to be able to identify the encoding therefore you must use values as defined by IANA.org. https://www.iana.org/assignments/character-sets/character-sets.xhtml

Returns:

Returns the encoding set on the field on success or a c-treeDB error on failure.

Calling **ctdbSetFieldStringEncoding()** on a field type that is not a string field fails with error **CTDBRET_INVTYPE**.

ctdbGetFieldStringEncoding

Get the encoding set on the field as a string.

Declaration

```
pTEXT ctdbDECL ctdbGetFieldStringEncoding( CTHANDLE Handle )
```

Description

- *Handle* [IN] - the associated field handle.

ctdbGetFieldStringEncoding() returns the encoding set on the field as a string or Null if no encoding was set or an error occurred. Check **ctdbGetError()**.

Calling **ctdbGetFieldStringEncoding()** on a field without an encoding set results in returning NULL.

Returns

c-treeDB error code

Availability

These methods are available in the following interfaces:

- c-treeDB C++ API
- c-treeDB .NET API
- Node.js
- c-treeDB Java (JTDB):

```
public void SetFieldStringEncoding(String encoding) throws CTException
public String GetFieldStringEncoding() throws CTException
```

11.7 Use Plug-ins and Run Anything Server-Side

FairCom's V12 Release introduces a brand new ability to extend their functionality with advanced modular capabilities. Using a new plug-in architecture advanced features can be quickly dropped in and independently configured plug-ins are provided as a shared object and companion configuration file. Each plug-in is usually self-contained in its own unique folder for organization. All plug-in configurations are collected under a single configuration folder.

Several plug-ins are provided by default. Provided plug-ins include:

1. HTTP web services
2. MQTT message services
3. REST API services
4. OPC communication protocols for Industrial IoT (IIoT) services
5. UA communication interface for IIoT services
6. PTC ThingsWorx® IIoT integration
7. PTC ThingWorx® AlwaysOn IIoT protocols

Provided plug-ins are not enabled by default to maximize security. FairCom-provided Plug-ins are securely implemented; however, they can increase the attack surface by possibly opening additional network ports and listening across alternative communication protocols.

Plug-ins are enabled as needed in your standard FairCom configuration file, *ctsrvr.cfg*. Each Plug-in is individually enabled with a configuration line that includes its configured name and shared object location.

Example

```
; Plugins
PLUGIN cthttpd;./web/cthttpd.dll
PLUGIN ctagent;./agent/ctagent.dll
```

The *ctsrvr.cfg* configuration file is now located in <*faircom*>*server**config* (this location is optional and existing locations are supported for full backward compatibility).

More information on the Plug-In architecture is available in these locations:

- FairCom Edge: *Supported Plug-ins* (https://docs.faircom.com/doc/*c-treeEDGE_DevelopmentGuide/SupportedPlug-ins.htm*)
- FairCom DB: *Plug-In Architecture* (https://docs.faircom.com/doc/*ctreeplus/Plug-InArchitecture.htm*)

c-tree Server Can Load Plug-In On-Demand after Server Has Started

To dynamically load a plug-in on demand after c-tree Server has started up, use the **ctadmn** utility or use the same `PLUGIN` configuration option syntax that you would use in *ctsrvr.cfg* in a call to **ctSETCFG()**.

Example 1 - ctadmn utility:

1. Select option 10. Change Server Settings
2. Select option 10 again. Change the specified configuration option
   Enter the configuration option and its value:

```
>> PLUGIN cthttpd;./web/cthttpd.dll
Successfully changed the configuration option.
```

Example 2 - API function call:

```
ctSETCFG(setcfgCONFIG_OPTION, "PLUGIN cthttpd;./web/cthttpd.dll");
```

Web Plug-In - Default linked_ace_server

The HTTP Plug-in (*cthttpd*, the Web Server Plug-in) supports connecting to a remote c-tree server. The engine to be used by the web server is configured by the `linked_ace_server` property in *cthttpd.json*. That option allows you to specify which c-tree server to use for the REST API, MQTT persistence, etc. The default `linked_ace_server` is NULL, which means *FAIRCOMS@localhost*. To change this default, change this property in *cthttpd.json*.

Because the default usage of *cthttpd* is by the HTTP Plug-in (the Web Server Plug-in loaded by a c-tree server), the plug-in framework already has the "caller" server name in the plug-in structure. So, instead of considering the default `linked_ace_server` as NULL, we now assume the local server name as default.

Note that if `linked_ace_server` is configured in *cthttpd.json*, it will overwrite this local server name as default.

11.8 Tag, Find, Move, and Cache Data Files More Easily

c-treeDB Database Dictionary Support for Table Marks

In V12 and later, a new concept has been implemented in c-treeDB that makes it faster and easier to find specific files in applications with a large quantity of files: *table marks*. A table mark is a mark (or flag) associated with the table when it is added to a database (so it's a mark in the database dictionary). We currently support up to 16 different marks on each table. These marks can be used to "brand" the table with some meaning.

At this time, the only table mark implemented is _DICT_VERMARK_TBL_APP_LIST, which signifies that the table is branded so as to simulate the effect of the `APP_NAME_LIST` keyword, but instead of being listed in *ctsrvr.cfg* it is marked in the database dictionary.

The **ctdbAddTableXtd()** function has been modified so that the first parameter, *Handle* (previously interpreted as a database handle), can now be a table handle. If it is a table handle, the database handle is derived from it, and the *dictionary mark* setting in the table handle is used when adding the table to the dictionary.

New API functions have been created to support this feature:

ctdbFindTableDictionaryMark()

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

ctdbSetDatabaseTableMarkFilter()

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

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

ctdbMoveTable() Function to Rename Table and Change Path

The **ctdbRenameTable()** function can rename a table but it does not provide for "renaming" the path (i.e., using "rename" to move the file to a different location, as is common in Operating System commands and c-tree RENIFIL).

To provide this feature, a new function has been added: **ctdbMoveTable()** (https://docs.faircom.com/doc**/ctreedb/ctdbmovetable.htm**) allows changing the path or the name or both of an existing table.

c-treeDB - Added Support for Session and Database Dictionary in Regular, Non-Superfiles

c-treeDB now has the capability to create session and database dictionaries as regular tables instead of superfiles. When using this feature, c-treeDB now creates a directory named the same as the superfile was named. The dictionary tables are created as regular files inside the directory.

- The session dictionary is created as *ctdbdict.fsd\ctdbdict.dat ctdbdict.fsd\ctdbdict.idx*
- The database dictionary is created as *<databasename>.fdd\ctdbdict.dat* and *<databasename>.fdd\ctdbdict.idx*

To use this new capability, call this function prior to creating the session or the database:

ctdbSetDictInSuperfile()

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

The handle must be a session handle when creating a session and a database handle when creating a database.

Allow Opening a Data File Even if its Record Update Callback Resource DLL Cannot Be Loaded

A c-tree data file that requires a record-update callback DLL would fail to open with error code **981** if the callback DLL could not be loaded. In some cases, it might be desirable to allow the data file to be opened in this situation. This is now permitted.

An extended file open mode has been added. It can be used to permit the data file to be opened for read-only access if its record-update callback DLL cannot be loaded. The new extended file mode is `ctXOPN_NORUCB_OK`. To use this mode when opening the file, call **SetXtdFileOpenMode(***ctXOPN_NORUCB_OK***)** before opening the file.

Note that the extended file mode set by **SetXtdFileOpenMode()** overwrites any previous extended file mode that was set for the connection. Upon return from the file open call, the extended file mode is reset to zero. This implies the normal usage of this function is to call it right before opening the file.

Note: Although the file open succeeds in this situation, the file open function sets sysiocod to **RUCBDLL_NOT_LOADED_COD**, and a message such as the following one is logged to *CTSTATUS.FCS* indicating the failed DLL load:

```
CTDLL_LOAD: Failed to load module ctuser.dll: The specified module could not be found.
```

11.9 Process All Files Forward and Backward

Frequently, the newest and most relevant data is appended at the end of data files, for example, application audit logs and other time series captured data. While indexes can be used for reverse traversal, reading records backwards from the end of the file toward the beginning is the fastest method to process this data.

FairCom DB has always been able to read records backwards using an index, or when resources are not enabled on a data file. For example, you can *create an index* (https://docs.faircom.com/doc/*ctreeplus/createindexfile.htm*) on *RECBYT* (https://docs.faircom.com/doc/*ctreeplus/30863.htm*) and use it to read records backwards in the order they are stored in the data file. This is fast, but it is even faster to read records directly out of a data file without using an index.

This enhancement allows FairCom DB to read records backwards directly out of a data file without using an index. It only applies to data files containing fixed-length records. Data files containing variable-length records still require an index to read records in reverse order.

Compatibility

- This enhancement works when the database runs as a server supporting one or more clients (*client-server mode* (https://docs.faircom.com/doc/*ctreeplus/30181.htm*)).
- It also works in *non-server single-user mode* (https://docs.faircom.com/doc/*ctreeplus/30179.htm*) and *non-server multi-user mode /doc/ctreeplus/30180.htm*.
- This new functionality does not affect the client library. Thus, existing applications can take advantage of this feature without recompiling or linking to a new version of the client library.
- You must install the latest version of the database server or recompile or link to the latest version of the database when using it in non-server mode.

Limitations

- This enhancement does not work in *non-server multi-threaded multi-user mode* (https://docs.faircom.com/doc/*ctreeplus/30183.htm*).
- Data files containing variable-length records still require an index to read records in reverse order.

Database Internals

Before this enhancement, FairCom DB did not allow reading a data file in reverse physical order if that file contained resource records. For example, calling **LSTREC()** on such a file would fail with error 48 (**FMOD_ERR**).

FairCom DB imposed this restriction because resource records can span more than one fixed-length record block in the data file. Thus, scanning backward through a data file could cause the record pointer to end up in the body of a resource record. This is not a problem when reading records in forward order because the start of the resource includes its length, which makes it easy for FairCom DB to skip over the resource record.

FairCom DB eliminated this limitation by creating a list of resource records that contains the offset and length of each one. This list is constructed on the first backward scan call. When a resource is added, deleted, or updated, the resource list is refreshed.

11.10 New APIs to Control Replication

- *Replication Manager User Guide* (https://docs.faircom.com/doc/*replication-manager/*)
- *Replication C++ API Developer's Guide* (https://docs.faircom.com/doc/*cpp-replication-api/*)
- *Replication JSON RPC API Developer's Guide* (https://docs.faircom.com/doc/*java-json-rpc-replication-api/*)
- *Replication Extensions* (https://docs.faircom.com/doc/*replication_extension/*)

11.11 ISAM Lock Functions Exposed in Java and .Net

FairCom has identified c-treeDB functionality that was not available to programmers using C++, c-treeDB Java (JTDB), and .NET. Modifications have been made to make the following features available:

- All current lock modes added to the .NET *Faircom.Isam.dll*.
- **LOKREC** and **LKISAM** function definitions added to the JCtree package.

- **lockCtData** and **lockIsam** functions added to the CtreeFile package and the supporting enum.

12. New Platforms

FairCom DB expands platform availability for maximum flexibility with modern development requirements.

Windows Visual Studio

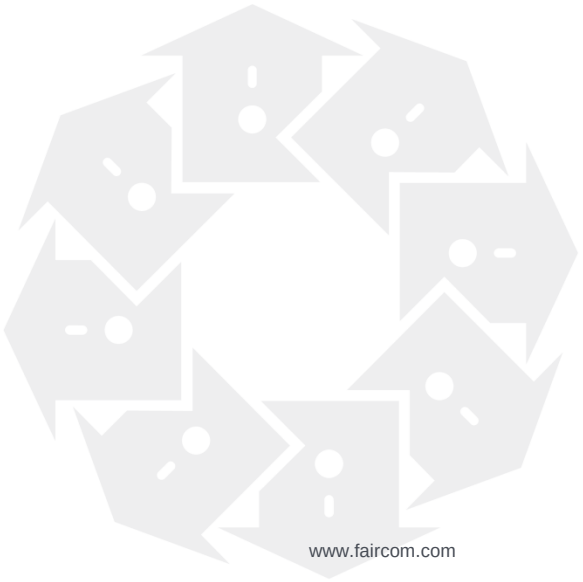
FairCom DB V12 supports the following Microsoft Visual Studio versions out of the box:

- VS 2015,
- VS 2017
- VS 2019

Prior VS versions remain available. Contact FairCom if you have a specific need for a legacy version.

New OS platforms

- ARM 32-bit and 64-bit
- Raspberry Pi OS
- macOS Catalina (V10.15)
- macOS Mojave (v10.14) and of course, most the prior versions are still supported if needed
- IBM Power 9 (P9)
- IBM s390 Mainframe under Linux - *Contact FairCom for availability*



13. Functions

The c-tree API set is your foundation for a successful application. Nearly every new FairCom DB feature starts with an underlying function. V12 introduces new API functions for the many new features and updated existing functions for extended and enhanced functionality. These changes span the entire stack of c-tree API functions. See how you can directly control your FairCom DB for the fastest possible performance.

13.1 c-treeDB Functions

JSON Handling

ctdbAddJSONSegment() (https://docs.faircom.com/doc**/ctreedb/ctdbaddjsonsegment.htm**) added to support JSON
indexing

ctdbGetFieldAsJSON() (https://docs.faircom.com/doc**/ctreedb/ctdbgetfieldasjson.htm**) added to return JSON fields from tables

ctdbSetFieldAsJSON() (https://docs.faircom.com/doc**/ctreedb/ctdbsetfieldasjson.htm**) added to retrieve a field as a CTJSON type value

ctdbSetJSONSegmentField()
(https://docs.faircom.com/doc**/ctreedb/ctdbsetjsonsegmentfield.htm**) added to modify underlying JSON field segment properties

Table Dictionary Marks

ctdbAddTableXtd() (https://docs.faircom.com/doc**/ctreedb/ctdbaddtablextd.htm**) extended to support table marks **ctdbFindTableDictionaryMark()**

(https://docs.faircom.com/doc**/ctreedb/ctdbfindtabledictionarymark.htm**) finds tables matching filter

ctdbSetDatabaseTableMarkFilter()

(https://docs.faircom.com/doc**/ctreedb/ctdbsetdatabasetablemarkfilter.htm**) sets a filter for easier subsequent search

ctdbSetTableDictionaryMark()
(https://docs.faircom.com/doc**/ctreedb/ctdbsettabledictionarymark.htm**) assigns a mask of 16 user-defined bitflags to a table

Field Masks

ctdbAddToFieldMask() (https://docs.faircom.com/doc**/ctreedb/ctdbaddtofieldmask.htm**) added for faster field return performance using masks

ctdbIsFieldMaskOn() (https://docs.faircom.com/doc**/ctreedb/ctdbisfieldmaskon.htm**) added to find if a field mask has been activated

ctdbRemoveFieldMask() (https://docs.faircom.com/doc**/ctreedb/ctdbremovefieldmask.htm**) added to empty and
deactivate a field Mask.

ctdbCheckFTIBackgroundLoad() added to check background FTI loading

ctdbCheckIndexBackgroundLoad()

(https://docs.faircom.com/doc**/ctreedb/ctdbCheckIndexBackgroundLoad.htm**) returns status of indexes loading in background

ctdbStartFTIBackgroundLoad() added to commence a background index load for a Full-Text Index (FTI)

Timestamp Handling

ctdbCurrentTimestamp() (https://docs.faircom.com/doc**/ctreedb/ctdbcurrenttimestamp.htm**) added to set current DATETIME with current time and millisecond precision

ctdbCurrentDateTimeUTC()

(https://docs.faircom.com/doc**/ctreedb/ctdbcurrentdatetimeutc.htm**) added to set a DATETIME field with current UTC time and millisecond precision

ctdbSetFieldAutoSysTime()
(https://docs.faircom.com/doc**/ctreedb/ctdbsetfieldautosystime.htm**) added to set automatic timestamps

ctdbTime*() and **ctdbStringToTime()**

(https://docs.faircom.com/doc**/ctreedb/ctdbstringtotime.htm**) functions support CTTIME_TYPE format string in hh:mm:ss:ttt format

Batches

ctdbGetFieldAutoSysTime()
(https://docs.faircom.com/doc**/ctreedb/ctdbgetfieldautosystime.htm**) gets auto timestamp status

ctdbSetBatchFilter() (https://docs.faircom.com/doc**/ctreedb/ctdbsetbatchfilter.htm**) added to set a batch filter
improving performance

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

added to remove record batch range

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

added to add record batch range

Embedded Server Functions

ctdbSetLogPath() (https://docs.faircom.com/doc**/ctreedb/ctdbsetlogpath.htm**) sets transaction log path for standalone applications

ctdbSetStatusLogPath() (https://docs.faircom.com/doc**/ctreedb/ctdbsetstatuslogpath.htm**) sets *CTSTATUS.FCS* path for standalone applications

ctdbSetTempPath() (https://docs.faircom.com/doc**/ctreedb/ctdbsettemppath.htm**) function to set the path for temporary files

Other c-treeDB Additions

ctdbGetFieldStringEncoding()
(https://docs.faircom.com/doc**/ctreedb/ctdbgetfieldstringencoding.htm**) added to obtain the the encoding set on the field as a string

ctdbSetDictInSuperfile() (https://docs.faircom.com/doc/**ctreedb/ctdbsetdictInsuperfile.htm**) specifies dictionary as multiple files or a single superfile

ctdbSetFieldAsBinary() (https://docs.faircom.com/doc/**ctreedb/ctdbsetfieldasbinary.htm**) enhanced with performance optimization in obtaining field buffers

ctdbSetFieldStringEncoding() (https://docs.faircom.com/doc/**ctreedb/ctdbsetfieldstringencoding.htm**) added to set a field string encoding

ctdbSetTablePartitionMaximumActive() (https://docs.faircom.com/doc/**ctreedb/ctdbsettablepartitionmaximumactive.htm**) purges expired partitions **ctdbTableHasCallback()**

(https://docs.faircom.com/doc/**ctreedb/ctdbtablehascallback.htm**) added to check of a table has any associated callback

ctdbTableHasLocks() (https://docs.faircom.com/doc/**ctreedb/ctdbtablehaslocks.htm**) added to check if a table has any active locks

ctdbTruncateTable() (https://docs.faircom.com/doc/**ctreedb/ctdbtruncatetable.htm**) added to remove all records in an existing table

13.2 ISAM Functions

Auto Timestamps

AddAutoSysTimeFields() (https://docs.faircom.com/doc/**ctreeplus/addautosystimefields.htm**) add auto-timestamp configuration to fields

RemoveAutoSysTimeFields() (https://docs.faircom.com/doc/**ctreeplus/removeautosystimefields.htm**) removes auto-timestamp configuration from fields

UpdateAutoSysTimeFields() (https://docs.faircom.com/doc/**ctreeplus/updateautosystimefields.htm**) updates existing auto-timestamp configuration to fields

WhichAutoSysTimeFields() (https://docs.faircom.com/doc/**ctreeplus/whichautosystimefields.htm**) retrieves auto-timestamp configuration information

JSON Handling

GetXtdKeySegmentDef() (https://docs.faircom.com/doc/**ctreeplus/getxtdkeysegmentdef.htm**) or **GETKSEGDEF()** (https://docs.faircom.com/doc/**ctreeplus/getxtdkeysegmentdef.htm**) enhanced to support JSON segments **PutXtdKeySegmentDef()**

(https://docs.faircom.com/doc/**ctreeplus/putxtdkeysegmentdef.htm**) or **PUTKSEGDEF()** (https://docs.faircom.com/doc/**ctreeplus/putxtdkeysegmentdef.htm**) enhanced to support JSON segments

Other c-tree API Additions

CreateIFileXtd8() (https://docs.faircom.com/doc/ctreeplus/createifilextd8.htm) extended to turn off *ctKEEPOPEN* mode for data file when connection closes (See Extended File Modes (https://docs.faircom.com/doc/ctreeplus/extendedfilemodes.htm))

ctSQLImportTable() (https://docs.faircom.com/doc/**ctreeplus/ctsqlimporttable.htm**) enhanced to optionally retain grants and synonyms when tables are removed from or restored into the SQL dictionary

ctSQLImportTable() (https://docs.faircom.com/doc**/ctreeplus/ctsqlimporttable.htm**) enhanced to enhanced to import data in UTF-8 and other iANA

ctTruncateFile() (https://docs.faircom.com/doc**/ctreeplus/cttruncatefile.htm**) immediately removes all data in a table, its data files and indexes

GetCtFileInfo() (https://docs.faircom.com/doc**/ctreeplus/getctfileinfo.htm**) or **GETFIL()** (https://docs.faircom.com/doc**/ctreeplus/getctfileinfo.htm**) retrieves maximum active files in a partitioned file **GetFileRegions()** (https://docs.faircom.com/doc**/ctreeplus/getfileregions.htm**) makes possible parallel file processing **SetXtdFileOpenMode()** (https://docs.faircom.com/doc**/ctreeplus/setxtdfileopenmode.htm**) extended to open files when update callback cannot be loaded

13.3 Low-Level Functions

LockCtData() (https://docs.faircom.com/doc/ctreeplus/lockctdata.htm) and LOKREC() (https://docs.faircom.com/doc**/ctreeplus/lockctdata.htm**)

- Enhanced to support *ctENABLE _BLKRQS* mode - promotes a non-blocking write lock request to a blocking lock
- Additional *ctCHKLOK_OTHER* mode indicates locks on a specified record offset
- Additional *ctFREE_REDLOKS* mode to free all read locks held by caller on the specified file

UpdateHeader() (https://docs.faircom.com/doc**/ctreeplus/updateheader.htm**) and **PUTHDR()** (https://docs.faircom.com/doc**/ctreeplus/updateheader.htm**) additions

- extended to assign a new unique file ID to a file using timestamp or transaction ID
- support setting maximum active partitioned files (https://docs.faircom.com/doc/ctreeplus/maxautopurgepartitions.htm) with *ctMAXPARTMBRhdr*
- extended to restrict ISAM key updates for calling connection only

UpdateRecordOffsetForKey()

(https://docs.faircom.com/doc**/ctreeplus/updaterecordoffsetforkey.htm**) updates record offset of specified key

ctVERIFYidx() (https://docs.faircom.com/doc**/ctreeplus/ctverifyidx.htm**) (and **ctvfyidx** (https://docs.faircom.com/doc**/ctreeplus/ctvfyidx-util.htm**) utility) updated to support searching for lost nodes **ctverifyFile()** (https://docs.faircom.com/doc**/ctreeplus/ctverifyfile.htm**) and **ctvfyfil** (https://docs.faircom.com/doc**/ctreeplus/ctvfyfil-util.htm**) updated to check validity of deleted space

13.4 System Functions

Plug In Callbacks

ctPlugin_describe() (https://docs.faircom.com/doc**/ctreeplus/ctplugin_describe.htm**) defines a set of callback functions loaded at server startup

ctPlugin_init() (https://docs.faircom.com/doc**/ctreeplus/ctplugin_init.htm**) initializes callback functions at server startup

ctPlugin_term() (https://docs.faircom.com/doc/**ctreeplus/ctplugin_term.htm**) deallocates call-back function resources at server shutdown

Embedded Server Configurations

ctSetConfigurationOptions()
(https://docs.faircom.com/doc/**ctreeplus/ctsetconfigurationoptions.htm**) passes in server configuration to embedded server models

ctSetLicenseFile() (https://docs.faircom.com/doc/**ctreeplus/ctsetlicensefile.htm**) sets a license file for an embedded server model

ctSetLicenseOptions() (https://docs.faircom.com/doc/**ctreeplus/ctsetlicenseoptions.htm**) passes license file contents to an embedded server model

ctSetLocalDirectory() (https://docs.faircom.com/doc/**ctreeplus/ctsetlocaldirectory.htm**) sets a local working directory location for an embedded server model

Security

SECURITY() (https://docs.faircom.com/doc/**ctreeplus/security.htm**) enhanced to validate user against master encryption password

ctGetAccountExpirationTime()
(https://docs.faircom.com/doc/**ctreeplus/ctgetaccountexpirationtime.htm**) outputs time remaining before account expires

High Availability Failover

ctSetClientLibraryOption()
(https://docs.faircom.com/doc/**ctreeplus/ctsetclientlibraryoption.htm**) enables additional c-tree client functions.

ctGetFailOverState() (https://docs.faircom.com/doc/**ctreeplus/ctgetfailoverstate.htm**) returns failover state variable **ctResetFailOverState()**

(https://docs.faircom.com/doc/**ctreeplus/ctresetfailoverstate.htm**) resets the failover state

Other

ctCopyFile() (https://docs.faircom.com/doc/**ctreeplus/ctcopyfile.htm**) allows opening files in shared mode if exclusive open fails

ctFILELIST() (https://docs.faircom.com/doc/**ctreeplus/ctfilelist.htm**) extended to client library support for KEEPOPEN
file list maintenance

ctSetCommProtocolOption()
(https://docs.faircom.com/doc/**ctreeplus/ctsetcommprotocoloption.htm**) set a client connection timeout in seconds

ctSysQueueOpen() (https://docs.faircom.com/doc/**ctreeplus/ctsysqueueopen.htm**) lists server system queues **SetOperationsState()**

(https://docs.faircom.com/doc/**ctreeplus/setoperationstate.htm**) or **SETOPTS()**
(https://docs.faircom.com/doc/**ctreeplus/setoperationstate.htm**) extended with new blocking write lock mode maintained only for next record modification

SetSystemConfigurationOption()
(https://docs.faircom.com/doc/**ctreeplus/setsystemconfigurationoption.htm**) or **ctSETCFG()**
(https://docs.faircom.com/doc/**ctreeplus/setsystemconfigurationoption.htm**) enhanced to ac-

USERINFOX() (https://docs.faircom.com/doc**/ctreeplus/userinfox.htm**) enhanced to track per connection statistics

13.5 SQL Functions

fc_purge_db() (https://docs.faircom.com/doc**/sqlops/fc_purge_db.htm**) built-in procedure purges tables that do not exist on disk from system tables

13.6 Direct SQL Functions

ctsqlGet*() - returns **CTSQL_NULLRESULT** when a NULL value is present

- All **ctsqlGet*()** functions are revised except **ctsqGetBinary()**, **ctsqlGetBytes()** and **ctsqlGetBlob()**.

ctsqlSetParameterAsString() (https://docs.faircom.com/doc**/dsql/ctsqlsetparameterasstring.htm**) - sets SQL parameters from strings

ctsqlSet*Parameter() - extended to accept any parameter type

- Various **ctsqlSet*Parameter()** functions have been extended to be used on any parameter type. Data conversion occurs immediately on the call so that possible conversion errors are immediately returned.

ctsqlSetPreserveCursor() (https://docs.faircom.com/doc**/dsql/ctsqlsetpreservecursor.htm**) - keeps cursor open when transaction ends

14. Configuration

The FairCom DB is extensively configurable. New and updated options are included in V12 and many of these are targeted for specific performance and capacity needs. Be sure to review your current configuration and see what V12 has to offer.

14.1 New ctsrvr.cfg Location and Default Additions

ctsrvr.cfg Moved to the New config Directory by Default

The *ctsrvr.cfg* file has been moved from the working directory (where **faircom.exe** is located) to the new *config* directory. This move centralizes all configuration files within a single directory for easier management. The server tries to load *ctsrvr.cfg* from the following directories:

<faircom>/config/ctsrvr.cfg

falling back to:
<faircom>/server/config/ctsrvr.cfg

and finally to:
<faircom>/server/ctsrvr.cfg

The configuration file can be placed in a directory of your choosing by using any of these procedures:

- Passing a command-line parameter, e.g.:
  `ctsrvr CTSRVR_CFG my_path/my_config_filename`
- Setting the `CTSRVR_CFG` environment variable
- Calling **ctdbSetConfigurationFile()** before calling **ctThrdInit()** for embedded server models (not supported on FairCom RTG)

All Plug-In Settings and Config Files in config Directory by Default

By default, all of the current plug-ins' settings files and configuration are loaded from the *config* directory. This applies to Replication Manager, web server, ThingWorx, OPC, and Automatic System Time/TimeStamp) settings files.

Best Practice: User-created plug-ins do not have to follow this practice, because they can be written to load their settings from wherever you want. However, for consistency and simplicity of administration, it is *strongly recommended* as a best practice to place all settings files in the *config* directory.

Password Security Options (Commented Out) in Default Server Configuration File

Password security options have been added to the default *ctsrvr.cfg* file. These options are commented such they do not affect your configuration. They are easily enabled by removing the semicolon ("`;`") at the beginning of the initial `SUBSYSYEM` line:

```
;SUBSYSTEM SECURITY PASSWORD {
     MINIMUM_LENGTH   8
     REQUIRED_CLASSES 3
     EXPIRE_IN_DAYS   180
}
```

Dynamic Configuration Changes Now Persisted

FairCom DB now has the ability to store configuration options enabled or modified at run-time. A new file in *FAIRCOM.FCS* is automatically created (even in an existing *FAIRCOM.FCS*) at V12/V3 server startup. Stored keywords overwrite any setting in *ctsrcr.cfg*.

See

COMPATIBILITY NO_CONFIG_PERSISTENCE (https://docs.faircom.com/doc/ctserver/compatibility-no-config-persistence-config.htm)

As of this release, the only supported keyword for this option is `READONLY SERVER`. More keywords will be added in the future.

14.2 Server Configuration Defaults - PAGE_SIZE 32768 and LOG_SPACE 1 GB

The defaults in the Server Configuration File (*ctsrvr.cfg*) have been modified to optimize performance on modern systems.

- `LOG_SPACE` has been increased from 120 MB to 1 GB

```
LOG_SPACE 1 GB
```

- `PAGE_SIZE` has been increase from 8192 to 32768

```
PAGE_SIZE 32768
```

Warning: Changing the `PAGE_SIZE` (https://docs.faircom.com/doc/`ctserver/page-size-config.htm`) is a maintenance task that should be done carefully and with a full reliable backup. Practice on a copy of your data and server folder until you are confident with the results. For procedures, see *Adjusting PAGE_SIZE* (https://docs.faircom.com/doc/*FairCom-Installation/AdjustingPAGE_SIZE.htm*) in the *FairCom Installation Guide*.

Note: A file created with a larger `PAGE_SIZE` cannot be opened by a server with a smaller `PAGE_SIZE`.

14.3 Core

CHECK_FILENAME_VALIDITY (https://docs.faircom.com/doc/ctserver/check-filename-valiidity-config.htm)
allow/disallow create or open of file with drive-relative path on Windows; default YES (not allowed)

CHECKLOCK_FILE (https://docs.faircom.com/doc/ctserver/checklock-file-config.htm) list files to enforce locks with *ctCHECKLOCK* mode

CLEANUP_ABLIST_ON_ABORT (https://docs.faircom.com/doc/ctserver/cleanup-ablist-on-abort-config.htm) abort node list entries cleaned up when transaction aborts

CTSTATUS_MASK MATCH_FILE_ID (https://docs.faircom.com/doc/ctserver/ctstatus-mask-config.htm) extended to
include "Reassigning file ID" messages

DISK_FULL_ACTION (https://docs.faircom.com/doc/ctserver/subsystem-disk-full-action-config.htm) allows unlimited MAXRUNTIME script execution

LEAF_NODE_READ_LOCK (https://docs.faircom.com/doc/ctserver/leaf-node-read-lock-config.htm) concurrent leaf node read scalability

LOCAL_CONNECTION_ONLY (https://docs.faircom.com/doc/ctserver/local-connection-only-config.htm) restricts
connections to localhost only

OPTIMIZE_FILE_OPEN (https://docs.faircom.com/doc/ctserver/optimize-file-open-config.htm)enables file open
performance optimizations

OPTIMIZE_FILE_CLOSE (https://docs.faircom.com/doc/ctserver/optimize-file-close-config.htm) enables file close
performance optimizations

PENDING_FILE_OPEN_RETRY_LIMIT (https://docs.faircom.com/doc/ctserver/pending-file-open-retry-limit-config.htm) file open close scalability

SYSTEM_FILE_ID_LIST (https://docs.faircom.com/doc/ctserver/system-file-id-list-config.htm) enables system file ID list management

DIAGNOSTICS SYSTEM_FILE_ID_LIST (https://docs.faircom.com/doc/ctserver/diagnostic-system-file-id-list-config.htm) provides additional file id list diagnostics

MAX_DFRIDX_LOGS (https://docs.faircom.com/doc/ctserver/max-dfridx-logs-config.htm) default

increased to 100 MAX_PREIMAGE_DATA (https://docs.faircom.com/doc/ctserver/max-

preimage-data-config.htm) specifies temporary storage of pending large transactions

MAX_PREIMAGE_SWAP (https://docs.faircom.com/doc/ctserver/max-preimage-swap-config.htm) specifies temporary storage of pending large transactions

MAX_REPL_LOGS (https://docs.faircom.com/doc/ctserver/max-repl-logs-config.htm) default in-

creased to 100 OPEN_FILES_ALERT_THRESHOLD

(https://docs.faircom.com/doc/ctserver/open-files-alert-threshold-config.htm) FairCom Server configuration option to log message when number of open files exceeds the specified value

PLUGIN_CALLBACK external user callbacks at server startup

STACK_DUMP Windows exception handler defaults to full dump creation

UNCSEG_KEYCOMPRESS (https://docs.faircom.com/doc/ctserver/uncseg-keycompress-config.htm) automatically
enables leading and padding key compression for all Unicode indexes (requires Unicode version of FairCom RTG, available upon request)

VSS_WRITER_QUIESCE_TIMEOUT (https://docs.faircom.com/doc/ctserver/vss-writer-quiesce-timeout-config.htm)
allows setting a default timeout value

14.4 SQL

SQL_SERVER_LOG_SIZE (https://docs.faircom.com/doc/sqlops/SQL_SERVER_LOG_SIZE.htm) limits *sql_server.log* size

SQL_SESSION_TIMEOUT (https://docs.faircom.com/doc/sqlops/sql-session-timeout-config.htm) sets a SQL Session Timeout Limit

SESSION_TIMEOUT (https://docs.faircom.com/doc/ctserver/session-timeout-config.htm) server keyword now affects SQL connections

SQL_TRACE_CTREE_ERROR (https://docs.faircom.com/doc/sqlops/sql-trace-ctree-error-config.htm) option now runtime configurable

SQL_OPTION DROP_TABLE_DICTIONARY_ONLY (https://docs.faircom.com/doc/sqlops/sql-option-drop-table-dictionary-only-config.htm) option to not physically delete
imported files on SQL DROP

SQL_OPTION BADDATES_ASNULL (https://docs.faircom.com/doc/sqlops/sql-option-baddates-asnull-config.htm) displays invalid dates as SQL NULL

SQL_OPTION BADTIMES_ASNULL (https://docs.faircom.com/doc/sqlops/sql-option-badtimes-asnull-config.htm)displays invalid time and timestamps as SQL NULL

SUBSYSTEM SQL LATTE MAX_STORE (https://docs.faircom.com/doc/sqlops/subsystem-sql-latte-config.htm) keyword for changing temporary store size limit

SYSLOG SQL_STATEMENTS (https://docs.faircom.com/doc/ctserver/syslog-config.htm) added detailed SQL statement logging to SYSLOG

SYSLOG USER_INFO (https://docs.faircom.com/doc/ctserver/syslog-config.htm) now logs SQL login/logout

SQL_DEBUG options are now dynamically configurable

SQL_DEBUG LOG_STMT (https://docs.faircom.com/doc/sqlops/sql-debug-log-stmt-config.htm) Improves SQL Statement Logging

SQL_DEBUG LOG_STMT_TIMES (https://docs.faircom.com/doc/sqlops/sql-debug-log-stmt-times-config.htm) include time spent in TEMP/SORT tables

SQL_DEBUG TRACE_ON_PANIC (https://docs.faircom.com/doc/sqlops/sql-debug-trace-on-panic-config.htm) option to create stack trace on PANIC

14.5 COMPATIBILITY

COMPATIBILITY COPY_FILE_OPEN_SHARED (https://docs.faircom.com/doc/ctserver/compatibility-copy-file-open-shared-config.htm) re-stores file copy behavior of opening files in shared mode

COMPATIBILITY KEEP_EMPTY_NODE_ON_READ (https://docs.faircom.com/doc/ctserver/compatibility-keep-empty-node-on-read-config.htm) pre-vents empty nodes found during reads from disk being added to the delete queue (returns to

V11 and earlier behavior) COMPATIBILITY NO_CONFIG_PERSISTENCE (https://docs.faircom.com/doc/ctserver/compatibility-no-config-persistence-config.htm) do not persist configuration changes done at runtime

COMPATIBILITY SQLIMPORT_ADMIN_PASSWORD (https://docs.faircom.com/doc/ctserver/compatibility-sqlimport-admin-password-config.htm) re-stores prior ADMIN
password usage in clear on command line

14.6 DIAGNOSTIC

DIAGNOSTICS CHECK_KEY_MARKS (https://docs.faircom.com/doc/ctserver/diagnostics-check-key-marks-config.htm) for unexpected key marks

DIAGNOSTICS FALG_ERR (https://docs.faircom.com/doc/ctserver/diagnostics-falg-err-config.htm) added to diagnose **FALG_ERR**

DIAGNOSTICS FILE_CLOSE_FLUSH (https://docs.faircom.com/doc/ctserver/diagnostics-file-close-flush-config.htm) turns on diagnostic logging of flush during file close

DIAGNOSTICS LOWL_FILE_IO (https://docs.faircom.com/doc/ctserver/diagnostics-lowl-file-io-config.htm) corrected to output messages for missing files

DIAGNOSTICS MEMTRACK (https://docs.faircom.com/doc/ctserver/diagnostics-memtrack-

config.htm) is now deprecated DIAGNOSTICS NODEQ_MESSAGE

(https://docs.faircom.com/doc/ctserver/diagnostics-nodeq-message-config.htm)
improved detail of delete node queue diagnostics

DIAGNOSTICS POPN_ERR (https://docs.faircom.com/doc/ctserver/diagnostics-popn-err-config.htm) outputs stack trace diagnostics on pending open error

DIAGNOSTICS READ_ERR (https://docs.faircom.com/doc/ctserver/diagnostics-read-err-config.htm) functionality

extended for standalone use

DIAGNOSTICS UPDFLG (https://docs.faircom.com/doc/ctserver/diagnostics-updflg-config.htm) identifies when a file is marked corrupt with additional point of occurrence logging

15. Utilities

Many existing utilities have been updated. This section describes scriptable command utilities. Be sure to check out new browser-based tools (page 48) as well!

15.1 Data Replication

ctrepd (**https://docs.faircom.com/docs/en/UUID-b16dee10-4a72-a00b-c432-aa54328e430c.html**) enhanced with additional options for file and path management

15.2 Backup and Restore

ctdump (https://docs.faircom.com/doc**/ctserver/ctdump-util.htm**) utility has been enhanced to support output to system stdout.

ctrdmp (https://docs.faircom.com/doc**/ctserver/ctrdmp-util.htm**) utility has been enhanced to support system input from stdin.

15.3 Data and Index File Management

- **ctstat** (https://docs.faircom.com/doc/ctreeplus/ctstat-util.htm)added file open/close stats added and SnapShot counter structures updated
- **ctcmpcif** (https://docs.faircom.com/doc**/ctreeplus/ctcmpcif-util.htm**)utility enhanced with option reducing page size of variable-length data files
- **ctpartadmin** (https://docs.faircom.com/doc**/ctreeplus/ctpartadmin-util.htm**) utility manages partition files: add, delete, archive and rebuild partitions and return partition status
- **ctvfyfil** (https://docs.faircom.com/doc**/ctreeplus/ctvfyfil-util.htm**) utility enhanced with option to check validity of deleted space in data files
- **ctvfyidx** (https://docs.faircom.com/doc**/ctreeplus/ctvfyidx-util.htm**) utility enhanced with option to check for lost index space
- **ctdmpidx** (https://docs.faircom.com/doc**/ctreeplus/ctdmpidx-util.htm**) utility enhanced to check for unexpected transaction marks
- **ctfileid** (https://docs.faircom.com/doc**/ctreeplus/ctfileid-util.htm**) enhanced to support reassigning a transaction controlled file's ID
- **ctmtap** (https://docs.faircom.com/doc**/ctreeplus/ctmtap-util.htm**) utility enhanced to run performance tests for specified duration rather than specific number of transactions

15.4 Caching

ctstat (https://docs.faircom.com/doc/ctreeplus/ctstat-util.htm) statistics utility now provides list of files on specific internal server lists, such as *ctKEEPOPEN*.

ctclosefile (https://docs.faircom.com/doc**/ctreeplus/ctclosefile_util.htm**) utility added with the following features

- Close files on server KEEPOPEN _LIST held open by server
- Adds, removes and closes files on server KEEPOPEN_LIST

15.5 Security

SQL utilities enhanced with TLS/SSL support using *[-S BASIC|<cert filename>]* command-line option to force a TLS connection.

- **isql** (https://docs.faircom.com/doc**/isql/**)
- **dbdump** (https://docs.faircom.com/doc**/isql/dbdump_util.htm**)
- **dbschema** (https://docs.faircom.com/doc**/isql/dbschema_util.htm**)
- **dbload** (https://docs.faircom.com/doc**/isql/dbload_util.htm**)
- **dbdeploy** (https://docs.faircom.com/doc**/cmdline/dbdeploy-util.htm**)

sa_admin (https://docs.faircom.com/doc**/ctserver/sa_admin-util.htm**) utility enhanced to display file permissions. **ctstat** (https://docs.faircom.com/doc**/ctreeplus/ctstat-util.htm**) utility enhanced to track user information for each connection.

15.6 Logging and Recovery

- **ctstat** (https://docs.faircom.com/doc**/ctreeplus/ctstat-util.htm**) utility enhanced to display log save time delta values
- **ctldmp** (https://docs.faircom.com/doc**/ctreeplus/ctldmp-util.htm**) utility enhanced to display file ID values
- **ctrdmp** (https://docs.faircom.com/doc**/ctserver/ctrdmp-util.htm**) utility extended with !RECOVER_DETAILS and !DIAGNOSTICS TRAN_RECOVERY
- **ctfdmp** (https://docs.faircom.com/doc**/ctreeplus/ctfdmp-util.htm**) utility extended with !RECOVER_DETAILS option for progress notifications

15.7 SQL Import

ctsqlimp (https://docs.faircom.com/doc**/sqlops/ctsqlimp-util.htm**) utility enhanced to import data in UTF-8 and other iANA assigned character sets (requires Unicode version of the server, available upon request).

15.8 Diagnostic Session Recording

cttrap (https://docs.faircom.com/doc**/ctreeplus/cttrap-util.htm**) utility supports displaying TRAPCOMM log contents.

15.9 FairComConfig Utility Moved

The **FairComConfig.exe** utility has been provided in past releases as a way of setting up the Windows service and making other registry changes. It is used by people who install from the Zip file. It is not required for people installing from the *.MSI*.

In this release it has been moved to *<faircom>\tools\SetUp*.

This utility is available only on Windows.

16. Callbacks for Custom Behaviors

A particularly useful feature FairCom application developers is the ability to embed custom application logic directly into the FairCom database. Callbacks are a programming technique allowing a developer to hand off processing directly at runtime. Callbacks provide a powerful method to introduce alternate logic for highly customized solutions. FairCom allows developers to hook into c-tree at many different levels for advanced precision control.

A new document lists available callback hooks throughout the FairCom Database ecosystem: **FairCom Callbacks** (https://docs.faircom.com/doc**/faircom-callbacks/**)

17. Upgrading and Application Building

Note: Several of the enhancements in this release require-client side upgrades. Our best practice is to always match the client-side to the same version of the Server being used.

17.1 Steps to Upgrade FairCom Server

While FairCom always attempts to maintain backward compatibility whenever possible, transaction logs from earlier versions are generally not always compatible with newer FairCom Server formats.

Note: Unless otherwise mentioned in the version-specific Update Guides, existing data and index files are usually not affected by transaction log changes.

Removing Transaction Logs and Upgrading the FairCom Database Engine

The FairCom process makes it easy to upgrade the FairCom server using your existing files, as long as you remove prior transaction logs in a safe manner. The steps shown below are appropriate any time you are upgrading. Notice that you will shut down your server two times during this process (steps 2 and 5) to allow all files to be brought to a consistent state.

1. Have all clients cleanly exit from your existing FairCom server.
2. Perform a normal **controlled shutdown** of the server using one of the methods described here, depending upon your installation:
   - Server Console Window - From the FairCom Server console window click “Control” and then click “Shutdown the Server”
   - Windows Toolbar - Right-click the c-tree Server icon in the Windows Tooltray and choose “ShutDown the Server”
   - Windows Service - From the Windows Control Panel, choose “Administrative Tools”, then choose “Services”. Locate the **FairCom Server** in the list of services running on your machine. Right-click the c-tree Service and choose “Stop”.
   - Use the client command line utility, **ctadmn**, and follow the prompts.
   - Use the client command line utility, **ctstop**.

   Remember that the Administrator user ID is "admin" (case insensitive) and the default password is "ADMIN" (case sensitive). The default c-tree Server name is "FAIRCOMS".
3. Block the ability of any clients to attach to the FairCom server.
4. Restart the **existing** FairCom server with no clients attached and allow a successful automatic recovery to take place. This ensures all files are brought to a consistent state in the event there is any data remaining in the transaction logs.
5. Perform another normal **controlled shutdown** of the FairCom server as described earlier.
6. Remove all existing transaction logs and associated files (*L*.FCS*, *S*.FCS*, *D*.FCS*, *I*.FCS*, and **.FCT*).

> **Note**: We do not recommend removing *FAIRCOM.FCS* unless specifically instructed to do so in the notes accompanying the new FairCom version.

Several other types of log dependencies should be considered before deleting transaction logs:

1. Replicas
2. Deferred indexes
3. Record update callbacks

If these options are in use, ensure they have processed all the log data before the logs are deleted. Otherwise the dependent data could be out-of-sync. (2 & 3 don't always use the transaction logs)

7. Copy your new FairCom server directory in its entirety into the existing Server directory. Note you might want to protect your existing *ctsrvr.cfg* and *ctsrvr.set* files so you don't loose any custom settings. Also review the new *ctsrvr.cfg* file accompanying any FairCom server upgrade to leverage new best practice settings.

> **Note:** Client compatibility can prevent connections to the new FairCom Database Engine. It is always advised to use the most recent matching client version with your FairCom server version.

8. Unblock the ability of any clients to attach.
9. Start the FairCom server in your usual manner and begin using your existing data.

FairCom has added logic to notify you when transaction logs may be incompatible. Please review the section "*Detection of Transaction Log Incompatibilities*" in the *FairCom Server Administrator's Guide* (https://docs.faircom.com/doc/*ctserver/*) for details.

17.2 Compiling Your Application

FairCom DB includes **mtmake.exe** (**mtmake** on Linux/Unix) located in *\<your installation folder>\build_sdk\build* for building the Standalone libraries. Once you follow the prompts to build your library of choice, execute **mk.bat** (**mk** on Linux/Unix) to build your project. For more details, see **mtmake (https://docs.faircom.com/doc/knowledgebase/50158.htm).**

Include Files

The *include* files needed for linking your client applications have been consolidated into the following folders:

- *\<your installation folder>\build_sdk\include*
- *\<your installation folder>\build_sdk\build* *<your arbitrary directory specified in mtmake.exe>*

Libraries

The FairCom DB Standalone library to link with your application will be located in

\<your installation folder>\build_sdk\build\<your arbitrary directory specified in mtmake.exe>\obj\release

(or it will be located in *debug* if you choose the default when executing **mk.bat**)

Compiling Examples

For examples of how to compile your client application, consult the *ctree.mak* file created when you execute **mtmake.exe**.

Required Libraries

The following libraries are required when building C/C++ applications.

Linux:

OpenSSL 1.0.2t	SSL support needs to be installed on most Linux distros. The correct version of OpenSSL is included in our package and our make file points to it. To include the proper encryption system libraries, you will likely need to add the following link switches: *-lssl -lcrypto* You can review */drivers/c.isam/tutorials/cmdline/Makefile* for a working example of building C and C++ applications.
Libz	Compression support. needs to be installed on CentOS.
libncurses5	Text based interface is included on most Linux distros.
gcc, make	Compiling tutorials are included on most Linux distros.
glibc	Included in most Linux distros. If your glibc is before 2.17 you will also have to include the rt library (-lrt) for “clock_gettime”.

Windows:

MS Visual Studio Runtime	• user32.lib • gdi32.lib • winspool.lib • comdlg32.lib • advapi32.lib • shell32.lib • kernel32.lib • oldnames.lib • crypt32.lib

17.3 Increased PAGE_SIZE for Improved Performance

In this release the `PAGE_SIZE` has been increased from 8192 to 32767, yielding a 15% performance gain in our testing. This requires new transaction logs and rebuilding all existing indexes and superfiles.

Warning: Changing the `PAGE_SIZE` (https://docs.faircom.com/doc/`ctserver/page-size-config.htm`) is a
maintenance task that should be done carefully and with a full reliable backup. Practice on a copy of your data and server folder until you are confident with the results. For procedures, see *Adjusting PAGE_SIZE* (https://docs.faircom.com/doc/*FairCom-Installation/AdjustingPAGE_SIZE.htm*) in the *FairCom Installation Guide*.

Note: A file created with a larger `PAGE_SIZE` cannot be opened by a server with a smaller `PAGE_SIZE`.

17.4 Upgrade & Compatibility

Note: This upgrade REQUIRES new transaction logs for several new features, including Greater than 32K file handle support, new performance improvements, and new replication enhancements.

17.5 V12 Changes

With V12, FairCom has strengthened it's license file checks. Therefore, before rolling out a V12 production license (ctsrvr*.lic file), we recommend thorough testing on your production machine to ensure your license files are properly sized for your production hardware. Details on CPU counting and licensing can be found here (https://docs.faircom.com/doc/ctserver/CountingCPUs&Threads.htm).

FAIRCOM.FCS V9 and older must be recreated:

FairCom DB V12 and FairCom RTG V3 have deprecated an older algorithm that prevents usage of *FAIRCOM.FCS* files from FairCom DB V9 and older Server lines (and any customers using the *prev10logon* switch within their V10 and newer *ctsrvr*.lic* files). The solution is to recreate the *FAIRCOM.FCS* file by not moving this file forward.

Note: Existing user IDs and passwords will need to be recreated with V12 and V3.

See **Adjusting PAGE_SIZE (https://docs.faircom.com/doc/FairCom-Installation/AdjustingPAGE_SIZE.htm).**

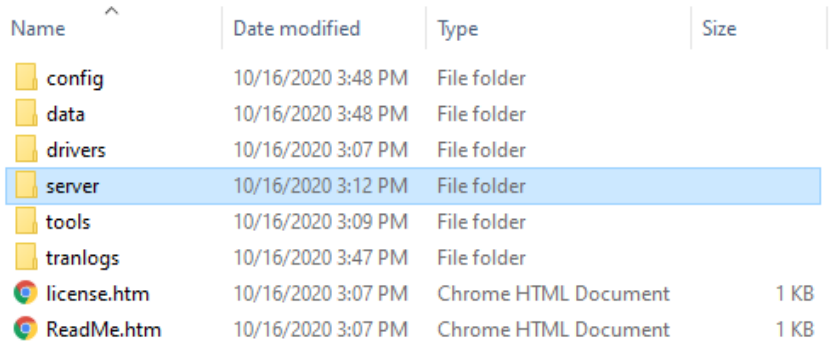
18. Compatibility Notes

With any new release, certain features preclude prior default functionality. FairCom DB V12 introduces several new advances that may impact your prior c-tree experience. Notably in this release is a new default folder layout (page 126) simplifying access to c-tree components. A major change to default IFIL path handling (page 130) is also included. Be sure to review this entire list and verify any specific changes that apply to your usage.

As always, V12 is a major version release change. It is expected to thoroughly test existing application functionality before deploying a new server version and to follow all recommended upgrade procedures (page 125) as described. Contact your local FairCom support team with any questions about compatibility and suggested upgrade steps.

18.1 Upgrade and New Default Folder Layout

A simpler organization of the FairCom folders is introduced to ease development productivity. It is a much flatter structure. (Yes, we were listening!) No more deep directory traversing. Everything is immediately at your fingertips — no more searching multiple folder hierarchies. You will find we renamed a few common folders, and included several new default folders. The descriptions below will guide you through.



Name	Date modified	Type	Size
config	10/16/2020 3:48 PM	File folder	
data	10/16/2020 3:48 PM	File folder	
drivers	10/16/2020 3:07 PM	File folder	
server	10/16/2020 3:12 PM	File folder	
tools	10/16/2020 3:09 PM	File folder	
tranlogs	10/16/2020 3:47 PM	File folder	
license.htm	10/16/2020 3:07 PM	Chrome HTML Document	1 KB
ReadMe.htm	10/16/2020 3:07 PM	Chrome HTML Document	1 KB

config

All server configurations are now located in a single convenient location. Our new server-side Plugins feature uses a single JSON-based configuration for each module. This is also the new default location for *ctsrvr.cfg,* (page 129) the master server configuration file. Of course, you can keep your original configuration file location as we'll look there if this new directory isn't present. And, don't forget, you've always had dynamic options (https://docs.faircom.com/doc/ctserver/alternative-server-configurations.htm) at server startup with environment variables and command line arguments.

data

This is the default server working directory you've always configured. We only moved it up a level. By default, you will find your newly created data and index files in this location. This is always configurable with the LOCAL_DIRECTORY (https://docs.faircom.com/doc/ctserver/local-directory-config.htm) server configuration. It is always recommended to locate this on a large fast volume to hold all of your application data.

drivers

Encompasses all FairCom DB drivers and APIs. You will find our complete assortment of all currently available drivers - 33 separate interfaces! -- and ready to compile and run tutorials for each. Look for new REST (page 89), node.js (page 95) (SQL and direct record oriented) and Python drivers here. This is equivalent to the */sdk* folder in prior releases.

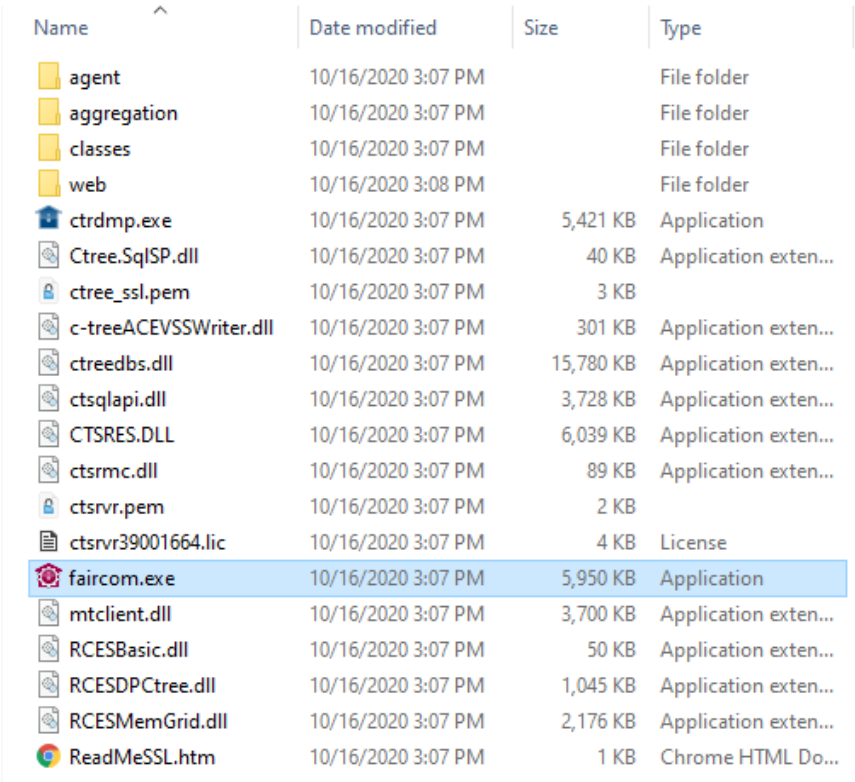
Inside this area we also have consolidated prior pre-compiled libraries, necessary project includes and binaries. These were previously found in */lib* and */bin* folders.

The embeddable server model (page 33) ("server dll") build area is found here as well. Look in the *ctree.srvdll* directory.

Another new feature in drivers is callback build support. Current support includes replication extension callbacks and master key retrieval for custom key storage solutions.

server

This is the same FairCom c-tree server you have always depended on. Just no more *bin/ace/sql* folder paths. Notice there are a few new folders inside for new Plugin (page 101) features. Data aggregation (page 48), replication agent, and the web apps (page 48) supporting our new browser based tools are located here. Each is activated with a specific Plugin configuration.



Name	Date modified	Size	Type
agent	10/16/2020 3:07 PM		File folder
aggregation	10/16/2020 3:07 PM		File folder
classes	10/16/2020 3:07 PM		File folder
web	10/16/2020 3:08 PM		File folder
ctrdmp.exe	10/16/2020 3:07 PM	5,421 KB	Application
Ctree.SqlSP.dll	10/16/2020 3:07 PM	40 KB	Application exten...
ctree_ssl.pem	10/16/2020 3:07 PM	3 KB	
c-treeACEVSSWriter.dll	10/16/2020 3:07 PM	301 KB	Application exten...
ctreedbs.dll	10/16/2020 3:07 PM	15,780 KB	Application exten...
ctsqlapi.dll	10/16/2020 3:07 PM	3,728 KB	Application exten...
CTSRES.DLL	10/16/2020 3:07 PM	6,039 KB	Application exten...
ctsrmc.dll	10/16/2020 3:07 PM	89 KB	Application exten...
ctsrvr.pem	10/16/2020 3:07 PM	2 KB	
ctsrvr39001664.lic	10/16/2020 3:07 PM	4 KB	License
faircom.exe	10/16/2020 3:07 PM	5,950 KB	Application
mtclient.dll	10/16/2020 3:07 PM	3,700 KB	Application exten...
RCESBasic.dll	10/16/2020 3:07 PM	50 KB	Application exten...
RCESDPCtree.dll	10/16/2020 3:07 PM	1,045 KB	Application exten...
RCESMemGrid.dll	10/16/2020 3:07 PM	2,176 KB	Application exten...
ReadMeSSL.htm	10/16/2020 3:07 PM	1 KB	Chrome HTML Do...

tools

Here are utilities and traditional visual tools. However, be sure to try out our new browser-based tools (page 48)!

Both the original .NET and Java tools are found in this directory, as well as a full set of command line client based utilities.

Windows Developers: Notice the **FairComConfig.exe** utility (for registering the **faircom.exe** Server as a Windows Service and for registering ODBC and ADO.NET drivers) has been moved to the *tools\setup* folder.

tranlogs

The FairCom Database Engine has always been able to locate transaction logs on a separate, independent (and your absolute fastest) storage volume. Since we've exhausted ourselves recommending this technique in training, we decided to make it the default, demonstrating the fullest potential of FairCom DB features. This is easily configured in your server configuration. Look for the default LOG_EVEN (https://docs.faircom.com/doc/ctserver/log-even-config.htm), LOG_ODD (https://docs.faircom.com/doc/ctserver/log-odd-config.htm)configurations now present.

ctsrvr.cfg Moved to New config Folder

The *ctsrvr.cfg* file has been moved from the working directory (where **faircom.exe** is located) to the new *config* directory. This move centralizes all configuration files within a single directory for easier management. The server tries to load *ctsrvr.cfg* from the following directories:

<faircom>/config/ctsrvr.cfg

falling back to:
<faircom>/server/config/ctsrvr.cfg

and finally to:
<faircom>/server/ctsrvr.cfg

The configuration file can be placed in a directory of your choosing by using any of these procedures:

- Passing a command-line parameter, e.g.:
  `ctsrvr CTSRVR_CFG my_path/my_config_filename`
- Setting the `CTSRVR_CFG` environment variable
- Calling **ctdbSetConfigurationFile()** before calling **ctThrdInit()** for embedded server models (not supported on FairCom RTG)

18.2 Increased PAGE_SIZE for Improved Performance

In this release the `PAGE_SIZE` has been increased from 8192 to 32767, yielding a 15% performance gain in our testing. This requires new transaction logs and rebuilding all existing indexes and superfiles.

Warning: Changing the `PAGE_SIZE` (https://docs.faircom.com/doc`/ctserver/page-size-config.htm`) is a
maintenance task that should be done carefully and with a full reliable backup. Practice on a copy of your data and server folder until you are confident with the results. For procedures, see *Adjusting PAGE_SIZE* (https://docs.faircom.com/doc*/FairCom-Installation/AdjustingPAGE_SIZE.htm*) in the *FairCom Installation Guide*.

Note: A file created with a larger `PAGE_SIZE` cannot be opened by a server with a smaller `PAGE_SIZE`.

18.3 Support Opening More Than 32,767 Files Affects Compatibility

FairCom DB V12 and FairCom RTG V3 clients are not compatible with servers from prior releases due to this new support. Transaction logs created by a server without 4-byte file number support are incompatible with servers that use 4-byte file number support, and vice-versa. When this incompatibility is detected, the database engine fails to start up with error **LFRM_ERR** (666), "incompatible log format."

For more information, see *Support opening more than 32,767 files (page 4).*

18.4 Max Replication and Deferred Index Logs Raised

The following keywords have been increased from the prior default of 50, to 100:

```
MAX_REPL_LOGS
MAX_DFRIDX_LOGS
```

Be aware that this change requires more disk space.

18.5 Improved IFIL Path Handling

Historically, the IFIL resource stored in the file includes a path component in *IFIL.pfilnam* and optionally *IIDX.aidxnam*, which could be absolute or relative. If the file is moved to a different directory, a mismatch exists between the path the application must specify to open the file and the path contained in the IFIL resource. This mismatch, particularly in the *aidxnam*, can cause problems, as this on-disk version of *aidxnam* is used by **OPNRFIL()** to locate these indexes.

Unless *SUPPRESS_PATH_IN_IFIL NO* is set in *ctsrvr.cfg*, the following changes take effect::

- The file becomes incompatible with older servers. This change takes effect when the **PUTIFIIL()** function is called, which typically occurs if a data or index definition changes. **PUTIFIL()** might be called automatically within operations such as file creation, an **AlterTable** operation, or potentially within a Rebuild or Compact operation.
- Applications may now specify *IIDX.aidxnam* as *"+myaltindex.idx"*, meaning that the alternate index is expected to be in the same directory as the data file. This syntax was previously available to **PRMIIDX()** and **TMPIIDX()** only. It is now available with all calls that accept an IFIL argument (**CREIFIL**, **OPNIFIL**, **RBLIFIL**, **CMPIFIL**, **RENIFIL**, **CLIFIL**, **DELIFIL**)
- Applications may now specify *IIDX.aidxnam* as *"?myaltindex.idx"*, meaning that the alternate index is expected to be in the same directory as the data file, and should be prefixed with the data file name as well: For *pfilnam="mypath/mydata"* and *aidxnam="?myaltindex.idx"*, the name of the index in the file system will be *"mypath/mydatamyaltindex.idx"*. ISAM-level renames of the data file will implicitly rename the alternate index.
- Beginning with V12, when a **PutIFile()** or one of it's extended versions is called, the path is automatically stripped off the file name *(IFIL.pfilnam*) and alternate index name *(IIDX.aidxnam*) elements of the IFIL structure.

These changes can cause compatibility issues. See these sections for information about disabling them:

- *Configuration option to disable IFIL path improvements* (page 131)
- *Programming option to disable IFIL path improvements* (page 131)

See also:

- *Partition file directory with SUPPRESS_PATH_IN_IFIL*

Configuration Option to Disable IFIL Path Improvements

New configuration option allows you to disable IFIL path improvements so files can be opened by earlier releases

FairCom DB includes enhancements to the storing of data and index file paths in the IFIL resource of a c-tree data file to avoid potential problems when files are moved to a different location. An attempt to open a data file that was created with these enhancements with a V11.5 (or earlier) server fails with error **FREL_ERR** (744, "file requires unavailable feature").

FairCom Server supports a configuration option that allows newly-created files to keep the IFIL resource in the old format, so that the data file can be opened by V11.5 and earlier. To use this option, add the following keyword to *ctsrvr.cfg*:

```
SUPPRESS_PATH_IN_IFIL NO
```

Note: This error will be seen only in environments in which a file is created on a server with the index file path enhancements and then is copied (replicated) to earlier versions of the server without this feature. As a best practice, FairCom always recommends keeping all servers in the same operating environment on the same release. If it is not possible to upgrade all servers to the same release, the keyword can be used as a last resort.

Programming Option to Disable IFIL Path Improvements

The new IFIL path logic can be deactivated programmatically using the following procedures.

Note: Programming support through the SDK only applies to standalone libraries. This technique is not supported in client-server applications.

The new IFIL path logic can be deactivated by setting the following environment variable after you have called **InitISAM()** and prior to creating any files:

```
ctSuppressPathInIFIL = NO;
```

For example:

```
    retval = INTISAMX(6,    /* index buffers */
            100,    /* files */
            64, /* page sectors => 8192 bytes cache page size */
            6,  /* data file buffers */
            MY_USER_PROFILE_MASK,   /* UserProfile */
            uid,    /* user id */
            upw,    /* user password */
            svn);   /* server name */
    if (retval) {
        printf("\nCould not initialize c-tree Plus(R) (%d)\n",retval);
#ifdef ctThrds
        ctThrdTerm();
#endif
        ctrt_exit(2);
    }

/* Setting this environment variable removes V11.6 and newer IFIL path logic */
ctSuppressPathInIFIL = NO;

        printf(" Attempting to create.\n");
            if ((retval = CREIFILX( &vcustomer, /* IFIL pointer */
                    " ",        /* data file name ext*/
                    ".ndx",     /* indx file name ext*/
                    (LONG) (OPF_ALL | GPF_READ | GPF_WRITE |
                    WPF_READ),  /* permission mask */
                    NULL,       /* alt group id */
                    NULL))       /* password */

                    || (retval = PUTDODA(CUSTDAT,doda,(UCOUNT) 7)))
            {
                printf("\nCould not create file with error %d (on file %d).\n",retval,isam_fil);
                fflush(stdout);
            }
            else
                printf("File created successfully.\n");
```

18.6 Shared Memory Performance Enhancement for all Unix Platforms

A shared memory performance enhancement has been enabled for all Unix platforms, starting with the base c-treeACE V11.6 line. The following changes have now been well proven in production use since late fall of 2017.

The Unix/Linux shared memory communication protocol has been changed to improve performance by improving the internal spin operation to be more efficient, especially for relatively short database operations.

Compatibility Note: It is important to recompile the client due to this shared memory change. A server that uses this modified shared memory protocol only supports shared memory connections from clients that also use this new enhanced protocol. If an older client (pre-V11.6) attempts to connect, it will fail with error **SHMC_ERR** (841) and the server will log the following message to *CTSTATUS.FCS*:

```
Fri May 26 12:13:07 2017
 - User# 00016  FSHAREMM: The client's shared memory version (3) is not
compatible with the server's shared memory version (4)
```

18.7 Increased Log Space Requirements

Several FairCom DB configurations have been modified, which will result in increased storage for transaction logs. Please note the following changes and configurations as applied to your environment.

LOG_SPACE

Total transaction log space has been increased to 1GB total space in default *ctsrvr.cfg* configuration files - `LOG_SPACE` (https://docs.faircom.com/doc/`ctserver/log-space-config.htm`). This means
each log now consumes 1/4, or 256MB of persisted storage space, including the log templates. As a result, your transaction log volume will now increase to 1GB on startup - three log templates and an active log. This will grow to 1.75GB when logs reach a steady state of default four log retention.

A change in logic allows us to avoid performing a long series of synchronous writes to the transaction log when we start the server in an empty transaction log directory.

This change significantly speeds up server startup on Linux when the file system has write barriers enabled.

Note: When write barriers are disabled on Linux, synchronous writes are much faster than with write barriers enabled, so FairCom highly recommends running with write barriers off and a battery backed power supply on the machine in production systems for best performance.

With this performance improvement, we have been able to increase `LOG_SPACE` to 1 GB and we have `COMPATIBILITY LOG_WRITETHRU` on by default.

The defaults in the Server Configuration File (*ctsrvr.cfg*) have been modified to optimize performance on modern systems.

- `LOG_SPACE` has been increased from 120 MB to 1 GB

```
LOG_SPACE 1 GB
```

- `PAGE_SIZE` has been increase from 8192 to 32768

```
PAGE_SIZE 32768
```

Warning: Changing the `PAGE_SIZE` (https://docs.faircom.com/doc`/ctserver/page-size-config.htm`) is a
maintenance task that should be done carefully and with a full reliable backup. Practice on a copy of your data and server folder until you are confident with the results. For procedures, see *Adjusting PAGE_SIZE* (https://docs.faircom.com/doc*/FairCom-Installation/AdjustingPAGE_SIZE.htm*) in the *FairCom Installation Guide*.

Note: A file created with a larger `PAGE_SIZE` cannot be opened by a server with a smaller `PAGE_SIZE`.

MAX_REPL_LOGS

MAX_REPL_LOGS (https://docs.faircom.com/doc/ctserver/max-repl-logs-config.htm) default has been increased to 100. This applies when replication is active and logs begin to accumulate when replication is
temporarily disconnected or falls behind. With the accompanying LOG_SPACE increase this can substantially increase persisted storage space requirements for transaction log volumes. That is, up to 100+ 256MB logs may be retained.

MAX_DFRIDX_LOGS

MAX_DFRIDX_LOGS (https://docs.faircom.com/doc/ctserver/max-dfridx-logs-config.htm) default has been increased to 100. This applies when deferred indexing is active and logs begin to accumulate when indexing is temporarily disabled or falls behind. With the accompanying LOG_SPACE increase this can
substantially increase persisted storage space requirements for transaction log volumes. That is, up to 100+ 256MB logs may be retained.

Tip - If you find error **96** on startup due to logs not found, it is likely be due to `MAX_DFRIDX_LOGS` or `MAX_REPL_LOGS` settings removing logs after the max is reached. Be sure to review your operational environment for appropriate settings.

18.8 Deprecated FairCom DB Configurations

DIAGNOSTICS MEMTRACK

Originally, enabling c-tree Server's memory allocation call stack collection required specifying `DIAGNOSTICS MEMTRACK` in *ctsrvr.cfg* because memory allocation call stack collection could increase memory use and slow performance.

An earlier modification made it possible for an administrator to enable and disable memory allocation call stack collection on a per-suballocator list basis at runtime, which meant that the collection was not necessarily a big impact on performance.

The `DIAGNOSTICS MEMTRACK` configuration option is now deprecated. This is now tracked internally without impact on performance. See the **ctstat** utility and **SnapShot()** function memory tracking, in addition to the new `HEAP_DEBUG_LEVEL` memory tracking.

LOCK_MONITOR

With file counter optimizations enabled for FairCom DB, the FairCom Server `LOCK_MONITOR` configuration is no longer available. Lock statistics available through **ctstat** and **SnapShot()** remain available and provide extended information than previously available.

18.9 Sort Module Error Code Changes

This modification replaces numerical error codes with explicit macros for error codes 471-497.

Notice that **JOBT_ERR** (471) collided with one of these errors, so sort error 471 has been changed to 476, which was an unused sort error code. The old sort errors 471-475 have been removed.

The new symbolic error names are as follows:

```
#define SORT_SWDEL_ERR  476 /* error deleting sortwork file */
#define SORT_ALC_ERR    477 /* error getting first data area */
#define SORT_INITS_ERR  478 /* sinit phase not previously performed-srelease */
#define SORT_RET_ERR    479 /* sreturn phase already started */
#define SORT_DATA_ERR   480 /* no records in data buffers */
#define SORT_INITR_ERR  481 /* sint phase not previously performed-sreturn */
#define SORT_NOMEM_ERR  482 /* not enough memory */
#define SORT_DATAP_ERR  483 /* no valid record pointers in merge buffers */
#define SORT_SWOPN_ERR  484 /* error opening sortwork file */
#define SORT_SWCRE_ERR  485 /* error creating sortwork.00x file */
#define SORT_SIZO_ERR   486 /* no records fit in output buffer */
#define SORT_READ_ERR   487 /* error reading sortwork file */
#define SORT_SIZM_ERR   488 /* bytes in buf <> merge buf size */
#define SORT_PTR_ERR    489 /* error adjusting file pointer */
#define SORT_SWECL_ERR  490 /* error closing sortwork.00x */
#define SORT_SWCL_ERR   491 /* error closing sortwork file */
#define SORT_SWDEL2_ERR 492 /* error deleting sortwork file */
#define SORT_REN_ERR    493 /* error renaming sortwork.00x */
#define SORT_CLSO_ERR   494 /* error closing output file */
#define SORT_CREO_ERR   495 /* error creating output file */
#define SORT_SWAP_ERR   496 /* insufficient disk space or no more work file segments */
#define SORT_PATH_ERR   497 /* ct_tmppth too long */
```

18.10 Default to IPv6 in Windows when TCP/IP is Selected

When the user selects TCP/IP, the FairCom build system now defaults to enabling IPv6 instead of only IPv4. This change makes this Internet protocol available by default to ensure your applications have support for the latest standards.

To begin using IPv6 support, you must enable the TCP/IP communication protocol support on the FairCom DB Server side by making sure this keyword is enabled in *ctsrvr.cfg*:

```
COMM_PROTOCOL           F_TCPIPV6
```

It is possible to have both IPv4 and IPv6 support enabled at the same time.

18.11 ctsqlGet*() Returns CTSQL_NULLRESULT when a NULL Value Is Present

Prior to this revision, a call to **ctsqlGetChar()**, or other similar functions for other data types, would fail with **SQL_ERR_BADARG** if the column value was a SQL NULL. These functions now return **CTSQL_NULLRESULT** if the data value is a SQL NULL. This change affects the following functions:

- **ctsqlGetBigInt**
- **ctsqlGetBit**
- **ctsqlGetChar**
- **ctsqlGetDate**
- **ctsqlGetFloat**
- **ctsqlGetInteger**
- **ctsqlGetMoney**
- **ctsqlGetNChar**
- **ctsqlGetNumeric**
- **ctsqlGetNumericAsString**
- **ctsqlGetReal**
- **ctsqlGetSmallInt**
- **ctsqlGetTime**
- **ctsqlGetTimeStamp**
- **ctsqlGetTinyInt**

It does not apply to **ctsqlGetBinary** or **ctsqlGetBlob**.

Note: This modification is a Compatibility Change.

18.12 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.

18.13 Better Error Reporting when Exceeding the Maximum Length of VARCHAR Fields

In versions prior to FairCom DB V12 and FairCom RTG V3, when ISAM inserts a string in a VARCHAR field that exceeds the maximum size allowed by SQL, the following occurs: an error is reported, the record is truncated, and a panic is logged in *CTSTATUS.FCS*. This is desirable behavior, but the error code does not help identify the error and offending rows are not identified by `Select * from table ctoption(badrec).`

In this release, a more helpful error code of "bad record" is returned, and the offending rows are identified by `Select * from table ctoption(badrec).`

For example, if ISAM updates a field with 10,000 characters when the maximum number of characters in SQL is 8,000, a "bad record" occurs, a panic is logged in *CTSTATUS.FCS*, and `Select * from table ctoption(badrec)` identifies the offending rows.

18.14 Disk Full Monitoring Keywords Added to Default ctsrvr.cfg

Disk Full Monitoring keywords have been added to the default Server config file, *ctsrvr.cfg*. These keywords are present in the config file, but they have been commented out for your convenience (so they are there if you need them, but they are NOT enabled).

```
; SUBSYSTEM EVENT DISK_FULL_ACTION {
    volume      VOLUME
    limit       LIMIT
    run         EXE [OPTIONS]
    freq        FREQUENCY
    maxruntime  MAXRUNTIME
}
```

18.15 SQL Statement Diagnostic Logging Keyword Added to Default Server Config

The SQL statement diagnostic logging keyword has been added to the default Server config file, *ctsrvr.cfg*:

```
; SQL statement diagnostic logging
;SQL_DEBUG   LOG_STMT
;SQL_SERVER_LOG_SIZE   100MB
;SETENV  TPE_LOG=<alternate path to sql_server.log>
```

This keyword is present in the config file, but it has been commented out for your convenience (so it is there if you need it, but it is NOT enabled).

18.16 Updated ctMAX_KEY_SEG Default from 16 to 32

The maximum number of key segments allowed by index has been increased from 16 to 32. This is a compile time limit. If you find a situation where more than 32 segments are needed, please contact FairCom for possible solutions.

18.17 MAX_REPL_LOGS and MAX_DFRIDX_LOGS Default Values Increased to 100

The default values for `MAX_REPL_LOGS` and `MAX_DFRIDX_LOGS` from 50 to 100. This may be overridden by setting these server configuration values.

18.18 Windows Drive-Relative Paths Deprecated

CHECK_FILENAME_VALIDITY keyword for backwards compatibility

Disallow create or open of file with drive-relative path

```
CHECK_FILENAME_VALIDITY YES | NO
```

FairCom DB V12 and FairCom RTG V3 forward no longer support “Drive-Relative” Paths by default. The purpose is to ensure predictable, safe, and secure behavior. and all files must be opened using an absolute file path. For backwards compatibility, you can restore prior behavior by adding the `CHECK_FILENAME_VALIDITY NO` configuration option to *ctsrvr.cfg*.

On Microsoft Windows,® FairCom uses the concept of a "Drive-Relative Path" as an easy way to specify a relative path to where the database is installed on a drive. The path includes a drive letter without a following backslash. For example, *C:mydirectory\test.dat* is a drive-relative path

because there is not a backslash after the drive specification (*C:*). On the other hand, *C:\mydirectory\test.dat* is *not* a drive-relative path; it is an absolute path. A drive-relative path is relative to whatever happens to be the current working directory of the database on that drive. A process may have a current working directory for each drive.

For example, if the working directory on C: is *C:\FairCom*, then the drive-relative path *C:mydirectory\test.dat* is equivalent to *C:\FairCom\mydirectory\test.dat* while the absolute path *C:\mydirectory\test.dat* is unaffected.

Drive-Relative Paths can cause behavior that is unpredictable, unsafe, or unsecure. Thus, the database now returns an error when a Drive-Relative Path is specified in a call to create or open a file.

Compatibility Notes: This modification breaks compatibility by disallowing certain file names that used to be allowed. *The preferred solution is to use a full path.* (A less preferable solution is to use the configuration option `CHECK_FILENAME_VALIDITY NO` in *ctsrvr.cfg*. This option can also be changed at runtime. In standalone mode, this behavior is controlled by a field in the *CTGVAR* structure, *ctCheckFileNameValidity*. Setting it to a non-zero value enables the check; setting it to zero disables the check.)

Index Node Prune Feature - Ability to deactivate node pruning

FairCom DB's delete-node thread prunes empty nodes from index files in the background. It densely packs key data for optimal index performance. To prevent index corruption, it runs during idle times and opens the index file exclusively, which prevents external processes from opening it.

To disable the "index node prune" feature, for example to allow an external process to open an index file, use the following keyword:

```
COMPATIBILITY NO_DELNOD_QUEUE
```

The above keyword prevents the delete-node thread from running, which prevents automatic index optimization, however, it allows external processes to open the index file.

Another keyword, `COMPATIBILITY KEEP_EMPTY_NODE_ON_READ`, prevents empty nodes read from disk being added to the delete queue. This is the V2/V11 and earlier behavior.

Compatibility Note: This change breaks compatibility.

18.19 Ports

The FairCom Server provides services over TCP/IP ports and shared memory. These services include APIs and browser-based applications for managing and exploring the Server.

Default FairCom Connection Strings

Protocol	Connection String
ISAM	`FAIRCOMS@localhost:5597`
iSQL	`isql -u ADMIN -p ADMIN 6597@localhost:ctreeSQL`
iSQL secure	`isql -u ADMIN -p ADMIN ssl:6597@localhost:ctreesql`
JDBC	`getConnection("jdbc:ctree://localhost:6597/ctreeSQL", "ADMIN", "ADMIN")`
Python SQL	`pyctree.connect(user='ADMIN',password='ADMIN',database='ctreeSQL',host='localhost',port=`
ADO.NET	`User ID=ADMIN;Password=ADMIN;database=ctreeSQL;server=localhost;port=6597`
PHP SQL	`ctsql_connect(":6597@localhost:ctreeSQL", "ADMIN", "ADMIN")`

Default Protocols, Ports, and Names

API	Protocol	Port or Name
SQL	TCP/IP	**6597**
JSON NAV API	WebSocket	**8081**
MQTT (optional)	WebSocket	**8081**
MQTT (primary)	TCP/IP	**1883**
ISAM & CTDB	TCP/IP	**5597**
ISAM & CTDB	Shared Memory	**FAIRCOMS**
REST	HTTPS over TCP/IP	**8443**
FairCom Web Apps	HTTPS over TCP/IP	**8443**
Replication Manager Web App	HTTPS over TCP/IP	**7000**

See Also:

- *FairCom Ports*
- Configuring the Application Server
- Configuring FairCom DB (https://docs.faircom.com/doc/ctserver/8570.htm)
- Connection Strings
  https://docs.faircom.com/doc/ctserver/connection-strings-attributes-defaults.htm

For example, when the FairCom Server is running and TCP/IP ports are not blocked, you can use a web browser to run the FairCom browser-based applications and a simple REST API via *https://localhost:8443* or the insecure HTTP protocol *http://localhost:8080/*

Troubleshooting Connections

If a port is not working, check the following:

1. Ensure the FairCom server is running.
2. Ensure your firewall is not blocking the port.
3. Check to see if another application is already using the port.

Tip: If the *localhost* domain name does not work, use the IP Address 127.0.0.1.

The FairCom Server provides a variety of services that require TCP/IP ports for communication. The table below lists these ports and explains the configuration file and keywords that affect them.

- To modify **cthttpd.json**, see Configuring the Application Server
- To modify **ctsrvr.cfg**, see Configuring FairCom DB - full URL (https://docs.faircom.com/doc/ctserver/8570.htm)

Note: If a port is not working, check your firewall to make sure it is not blocked. Also check to see if another application is using the same port.

The ports listed below are defaults. FairCom recommends using the defaults until you are familiar with the FairCom Server.

You may want to change ports for the following reasons:

1. Avoid colliding with a port used by an existing application
2. Avoid a port blocked by a firewall
3. Run multiple instances of the FairCom Server on the same computer.

Note: If you change ports, be sure to use ports that are not in use; otherwise, port conflicts will prevent you from being able to communicate with the FairCom Server.

Service	Port	Config File	Setting	Examples & Notes
FairCom Web Applications	8443	cthttpd.json	V12.5: "https_port": 8443 V12: "listening_https_port": 8443	https://localhost:8443/ https://localhost:8443/
MQTT Explorer - SSL	8443	cthttpd.json	V12.5: "https_port": 8443 V12: "listening_https_port": 8443	https://localhost:8443/mq/ https://localhost:8443/mq/ https://localhost:8443/mqttexplorer/ https://localhost:8443/mqttexplorer/
SQL Explorer - SSL	8443	cthttpd.json	V12.5: "https_port": 8443 V12: "listening_https_port": 8443	https://localhost:8443/sql/ https://localhost:8443/sql/ https://localhost:8443/sqlexplorer/ https://localhost:8443/sqlexplorer/
ACE Monitor - SSL	8443	cthttpd.json	V12.5: "https_port": 8443 V12: "listening_https_port": 8443	https://localhost:8443/monitor/ https://localhost:8443/monitor/ https://localhost:8443/acemonitor/ https://localhost:8443/acemonitor/
REST API - SSL	8443	cthttpd.json	V12.5: "https_port": 8443 V12: "listening_https_port": 8443	https://localhost:8443/ctree/api/v1/openapi https://localhost:8443/ctree/api/v1/openapi

Service	Port	Config File	Setting	Examples & Notes
Insecure HTTP Port	8080	cthttpd.json	V12.5: "http_port": 8080 V12: "listening_http_port": 8080	The apps and REST API above can use this port when a secure connection is NOT requir (e.g., in a development lab)
Replication Manager	7000	ctagent.json	"memphis_sql_port": 7000	FairCom's Replication Manager web applicat
MQTT	1883	cthttpd.json	V12.5: "mqtt_port": 1883 V12: "mqtt_listening_port": 1883	Publish and subscribe to MQTT messages
WebSocket for JSON NAV API & MQTT protocol	8081	cthttpd.json	V12.5: "websocket_port": 8081 V12: "mqtt_websocket_port": 8081	Used by FairCom's MQTT Explorer for MQT messages
c-treeDB, FairCom DB ISAM, FairCom Low-Level APIs	5597	ctsrvr.cfg	SERVER_PORT 5597	Used by FairCom's client driver to communic with server
SQL API	6597	ctsrvr.cfg	SQL_PORT 6597	Used by FairCom's Server for SQL communications
Node-RED	1880	settings.js	"uiPort": 1883	Node-RED is an open-source project that ca use the FairCom Server. It is not a FairCom product. Its configuration file, *settings.js*, is located in the folder where you installed Node-RED. **Caution**: By default, Node-RED communicates passwords in the clear. Node-RED can be secured.

FairCom ports are configured in configuration files located in *<faircom>/config*. You can change them using any text editor.

- Files with the *.json* extension must follow the syntax rules of JSON files.
- Files with the *.cfg* extension are FairCom's configuration files.
  - One property per line
  - Property name is followed by white space and the property value.
  - A semi-colon ("`;`") at the beginning of the line comments out the line.

As always, be security conscious with which database services are running and listening on ports. Less access is better. You can turn off services by disabling plug-ins.

Tip: If the *localhost* domain name does not work, use the IP Address 127.0.0.1.

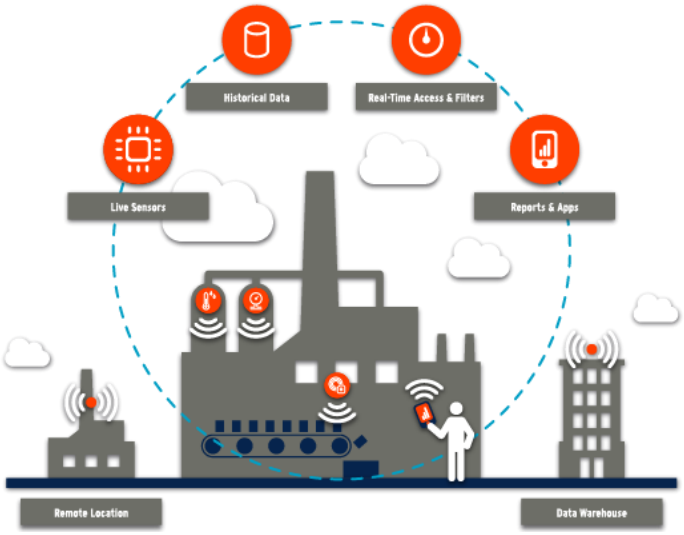
See Also:

- *Configuring Ports*
- *Connecting*

19. More FairCom Products

We'd like you to be aware of the following companion FairCom products that can compliment and work seamlessly with your FairCom Data.

19.1 FairCom Edge V3



FairCom Edge, an IoT focused product from FairCom first released in 2018, allows you to integrate anything in the factory with anything inside or outside of the factory. It is fine-tuned to run on the "edge"—close to data sources—in the Internet of Things. It allows you to connect, monitor, and control factory equipment, PLCs, hand-held devices, and sensors. FairCom Edge supports a variety of integration protocols, formats, and technologies to connect virtually any devices, equipment, and sensors to each other. It is designed as a foundation for modern, scalable, full-featured IoT solutions.

V3 features a growing list of interfaces, making it the ideal way to bridge the standards:

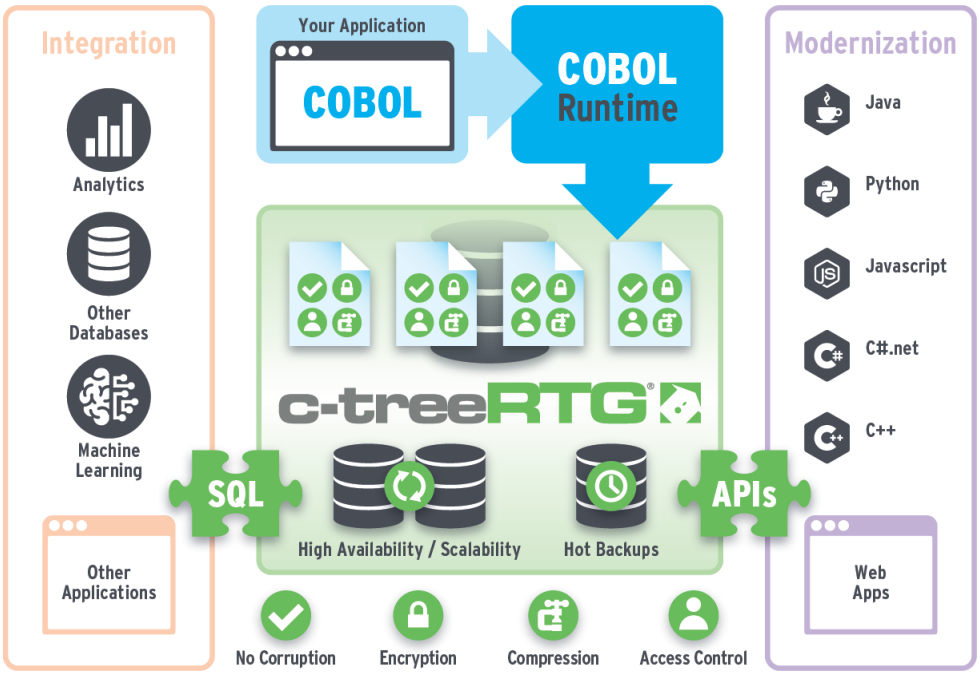
- ThingWorx "thing" supporting a REST API, MQTT, and the AlwaysOn protocol
- Node-RED Node
- Node.js for JavaScript integration
- MQTT Broker
- REST API supporting both navigational and relational (SQL) access
- OPC UA for industry-standard automation control

More information on FairCom Edge is available on our web site: FairCom Edge Product Page (http://www.faircom.com/products/c-treeedge-iot-database)

19.2 FairCom RTG V3

FairCom RTG COBOL Edition

COBOL remains a large and powerful force in daily transactions. Did you know about 95 percent of ATM card swipes are processed by COBOL code? COBOL powers nearly 80 percent of daily business transactions. More than 70 percent of mission-critical applications are authored in COBOL. *COBOL handles nearly $3 trillion in commerce every single day.* COBOL is here to stay for a very long time. Quite simply, it works!



FairCom RTG COBOL Edition leverages the technology of the FairCom DB Advanced Core Engine. FairCom RTG is designed as a direct replacement for your native COBOL file system and requires little or no modifications to your application. The FairCom RTG file system allows direct access to your files. By replacing your native COBOL file system with a specialized version of the FairCom Database Engine, FairCom RTG brings many benefits of FairCom DB directly to your COBOL applications, including:

- Transaction Processing
- Client/Server Architecture
- Sophisticated Caching
- Index Compression
- Multi-User Scalability and Performance
- Dynamic Hot Backups
- SQL Access to Your Data

19.3 FairCom RTG BTRV Edition

FairCom RTG BTRV Edition upgrades your legacy Btrieve applications by replacing the file system with the FairCom Database Engine. You obtain many of the same benefits of FairCom DB as described for FairCom RTG for COBOL without touching application code.

Copyright Notice

Trademarks

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.

11/12/2024

20. Index