test link

Update Guide

FairCom RTG V3 Update Guide

Audience

Developers and Architects

Subject

Using FairCom's FairCom RTG to modernize COBOL and BTRV applications with high-performance NAV and SQL technology

FairCom®

Contents

1. Introduction

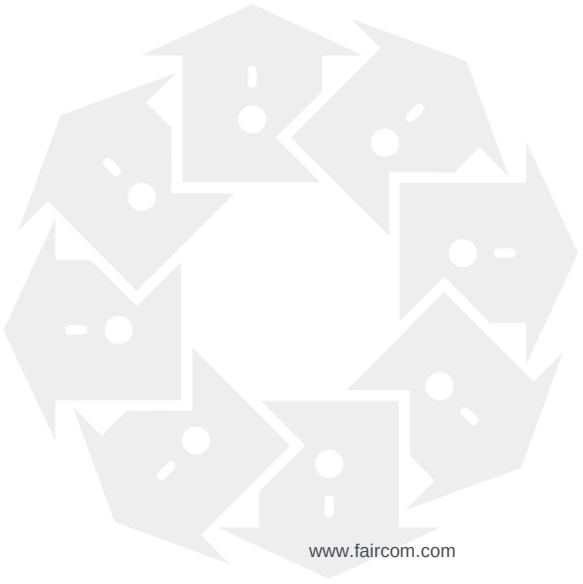
FairCom RTG V3 achieves ambitious new milestones in performance, security, and data migration while adding the latest version of FairCom Replication technology, Replication Manager, making it the perfect choice to modernize your COBOL applications to handle enterprise environments.

> *You may notice references to "FairCom DB," the database technology beneath FairCom RTG. Formerly called "c-treeACE" (Advanced Core Engine), this technology is the high-performance server that replaces your COBOL file system.*

The recommended upgrade procedures are described in the Compatibility Notes (page 106) chapter under **FairCom RTG V3 Upgrade & Compatibility** (page 100). Contact your local FairCom support team with any questions about compatibility and suggested upgrade steps.

In this Update Guide:

Note the changes discussed in this update guide go back to FairCom RTG V2. If you have received a FairCom RTG update after V2, it's likely some of the items in this guide may have already been communicated to you. All details in this book will be merged into the FairCom RTG online User's Guide, which should be your primary source of documentation once you have an understanding of what's new in FairCom RTG V3.



2. New FairCom RTG Features

The features and enhancements described in this chapter pertain specifically to FairCom RTG and are new in V3.

2.1 Expect Faster Application Throughput from Automatic Transaction Optimization

Transaction control is one of the most important data integrity benefits FairCom RTG brings to existing COBOL applications. Adding transaction support with FairCom RTG is transparent to the application with easy local configurations. Data integrity is provided by security data to persisted storage as soon as it is committed to the database. That persistence introduces a slight performance hit. Due to the nature of the automatic transaction support, this performance hit can be significant in various use cases. FairCom RTG V3 has improved this area of automatic transaction performance.

FairCom RTG creates tables as transaction processing capable by default; however, FairCom RTG applications seldom complete taking advantage of transaction processing control. COBOL, by default, opens tables in such a way that explicit transactions do not apply to them. In these cases, FairCom RTG now includes logic to detect and force automatic transaction processing.

When a FairCom RTG client sends a request for an update to the server, the following happens:

1. The server begins the transaction.
2. The server performs the update.
3. The server commits the transaction.
4. The server replies to client with the result.
5. The server waits for the next client request.

To enable this improved transaction optimization, include the following in your local FairCom RTG configuration options (*ctree.conf*). This is disabled by default.

This optimization reduces the amount of time before the server answers back to the client after an update. The server replies to the client while it is committing the transaction (the commit does not need to complete before returning back to the client). Compare the following sequence of events when enabled:

1. The server begins the transaction.
2. The server performs the update.
3. **The server replies to client with the result**
4. **The server commits the transaction.**
5. The server waits for the next client request.

By reducing the amount of time *before* returning the answer to the client, this greatly improves application throughput.

This has the effect that the record is not exactly committed or aborted by the time the client operation instigating the automatic transaction to the server (add, delete, update) returns. In case of disaster recovery, this may cause the last record update to be lost if the "disaster" occurred after the answer has been received by the client and before the commit is executed by the server, an expectedly very rare event.

This optimization is similar to the `DELAYED_DURABILITY` configuration (set on the Server side in *ctsrvr.cfg*) as both guarantee transaction atomicity and consistency and not durability, meaning the last transaction may be lost in rare cases of system failure. Typical envisioned failure scenarios include out of storage space, out of memory conditions, and storage system integrity issues. In case of transaction failure, the FairCom RTG server logs a *CTSTATUS.FCS* message and shuts down in a controlled coordinated effort.

2.2 Improved Delete Record Performance

This modification improves performance of delete record operations when preceded by a read record operation of the same record. The optimization of the delete function consists of avoiding the re-read of the record if the record's primary key was not changed since the last read operation.

The optimization is optional and controlled by a new local FairCom RTG configuration option `<delcurrent>` which is set to 'no' by default:

<delcurrent>

When deleting a record, avoid re-reading if the record's primary key was not changed since the last read operation.

Note: This option is effective only if:
1) The primary key of the file does not allow duplicates.
2) The record was locked during the read operation so it could not be replaced by another user between the read and delete operations.

Default: No.

2.3 Find Stuff Faster with Full-Text Search

This release introduces Full-Text Indexes for tables created through FairCom RTG COBOL and BTRV. The FairCom RTG table needs an internal ROWID (see below). Complete Full-Text Index documentation is available here: *Full-Text Search Guide* (https://docs.faircom.com/doc/*fts/*)

2.4 ROWID Record Header Required for Full Text Search

Full text search requires additional information on each record. A unique ROWID value provides this extended information. The new configuration element `<rowid>` enables this feature for newly created tables. ROWID is internally supported by the FairCom RTG serial segment feature (SRLSEG), an auto incrementing value transparent to the application.

ROWID support is enabled by including `<rowid>yes</rowid>` in your FairCom RTG local configuration file (*ctree.conf*).

1. Once you have created a table with ROWID, a Full-Text Index (FTI) can be created on the new table without further intervention.
2. Existing tables require a one-time conversion to support FTI. **ctutil** *-upgrade* enables this support.

The "size" attribute for the `<rowid>` tag can be used to set the size of the *rowid* hidden field in the record header. The default value is 8 for COBOL (4 for BTRV). Accepted values are 8 and 4.

```
<rowid size="8">yes</rowid>
```

Compatibility: The size setting of `<rowid>` is taken into consideration even if `<rowid>` is set to `no`, affecting other FairCom RTG features. Setting it to `8` (for COBOL, not setting a value has the same effect) makes new files incompatible with previous FairCom RTG versions.

Therefore, to be sure that files created with V3 are compatible with older versions, it is necessary to include the following in *ctree.conf*:

```
<rowid size="4">no</rowid>
```

2.5 Open up to One Million Concurrent Files in FairCom RTG

FairCom RTG Servers can now open and concurrently process up to one million files. Extending file numbering to a 32-bit value allowed this huge increase from 32,767. (We limited this support to 1,000,000 for practical reasons. A million files is a lot of files. Contact FairCom if you find a need for more.)

This allows FairCom RTG to be used in large-scale applications or in multi-tenant environments where a single FairCom RTG server can support hundreds of customers and applications with up to a total of 1 million open files per server. No changes are needed to take advantage of this new support, except for upgrading to V3 or later.

FairCom RTG clients remain limited to 32,767 files per connection, allowing compatibility with existing FairCom RTG clients. If an operation returns a file number out of this range, it fails and logs FairCom RTG error 3, "ERROR 3:0:0 too many files."

2.6 Faster Bulk Additions Speed Data Loads

Bulk addition is a great technique for quickly loading data into tables by allowing large blocks of records to be written into tables followed by a final index rebuild.

Bulk addition is enabled with the `<bulkaddition>` option in your local FairCom RTG *ctree.conf* configuration file. When `<bulkaddition>` is enabled, only data records are written and the keys later updated in one single index rebuild operation after the file is closed.

Previously the index rebuild routine used by FairCom RTG bulk addition scanned the data file for errors by default. In the case of bulk addition operations this data file scan is not necessary as indexes are only rebuilt to re-sync keys and not to find and fix file corruptions. Therefore, skipping the data scan phase of index rebuilds saves substantial time.

2.7 Faster Bulk Addition Index Rebuild from Cached Data

Bulk addition normally performs the index rebuild operation after the final data file is closed. At that point, all data is flushed from memory caches and resides in persisted storage requiring the rebuild phase to re-read the data, leading to increased I/O time.

An optional rebuild capability is available enabling the rebuild process to take advantage of data already existing in cache. In the best-case scenario, your entire data file fits in available FairCom RTG memory caches significantly reducing total rebuild time.

To enable cached rebuild performance add `<bulkaddition prmiidx>` to your local FairCom RTG *ctree.conf* configuration file. This option does not require closing the file and therefore takes advantage of the cached data when reading the data records and building keys.

2.8 Faster OPEN OUTPUT File Reuse

COBOL applications commonly delete all existing records in a file with a single `OPEN OUTPUT` operation that deletes and recreates an existing file. A new FairCom RTG optimization checks if the file being created already exists with the same file name and file definitions, and, if so, truncates existing data from the file rather than deleting and recreating a new file This saves several OS file operations improving responsiveness of the call.

To take advantage of this optimization, add `<truncateifexist>` to your local FairCom RTG *ctree.conf* configuration. This option is off by default.

If `<log><debug><file>` is also enabled, file truncation operations are logged with the message:

```
DEBUG FILE truncated: filename
```

2.9 Improved Detection of Logically Equivalent Filenames during Automatic Recovery

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

2.10 Standalone Operational Model Support

FairCom RTG is intended as a client/server architecture offering many advantages a single-server process can provide in total file coordination including transaction control, encryption, hot backups, etc.

However, by not going through the FairCom RTG Database Server, database I/O can at times be faster, typically when there is only a single instance performing database I/O. In addition, inter-process communication between client and server is eliminated. This model of access is termed *standalone*; the FairCom RTG COBOL process physically accesses files in the file system without reliance on a central server process (faircom.exe in V3 or ctreesql.exe in prior releases).

Possible use cases for standalone usage include high-speed data load (when loading a single file at a time – when loading multiple files concurrently, use of the FairCom RTG Database Server is still recommended), storing data locally on the application side for convenience, or for having local data storage that is only available to the client application, and therefore not shareable by other users.

Standalone support is enabled with the `<localinstance>` element of your local FairCom RTG *ctree.conf* configuration file. All `<file>` rules defined within a `<localinstance>` section cause matching files to be created/opened directly in the local machine (standalone model) rather than on the server environment.

Note: `<localinstance>` and (although not discussed here) `<redirinstance>` are only supported for the FairCom RTG BTRV interface and the COBOL ExtFH interface (Micro Focus).

<localinstance>

The <localinstance> element specifies instance-wide configurations for the standalone driver.

Note: **This element is only used for standalone versions of the product.**

Attributes

Attribute	Description	Default value
bufs	Specifies the number of index file cache buffers. Minimum of 3 required. Maximum is 32,767. Increasing these values can improve performance during bulk data loading.	1280
dbufs	Specifies the number of data file cache buffers. Minimum of 3 required. Maximum is 32,767. Increasing these values can improve performance during bulk data loading.	1280
fils	Specifies the initial block of file structures to allocate. Whenever the number of files required exceeds this initial amount, another block of file structures using this number is automatically allocated. Each index, whether it is a member of an index file or in a file by itself counts toward this parameter.	32
sect	Specifies the number of node sectors. Minimum of 1 required. This parameter multiplied by 128 equals the index node size. (The default of 256 = 32,768 bytes.)	256
logpath	Specifies the file path for the transaction processing log files.	""
temppath	Specifies the file path for storing temporary files.	""
sortmem	Specifies the size of sort buffers used by the local database instance.	100 MB

Example

```
<localinstance bufs="1000" fils="64" sect="128" dbufs="1000" logpath="/data/logs">
    ...
</localinstance>
```

Requirement

`<localinstance>` support requires the presence of a FairCom Standalone DLL or .SO, named *ctreestd.dll* or *ctreestd.so* for Linux/Unix systems. Contact FairCom for availability options.

Limitations

- Standalone usage does not support transactions. In essence, any update performed on a file defined under `<localinstance>` is committed immediately and cannot be rolled back. A "ROLLBACK TRANSACTION" operation is ignored and a warning message is logged.
- Standalone usage does not support `<runitlockdetect>`.
- Standalone usage does not support turning off `<optimisticadd>`, i.e., `<optimisticadd>no</optimisticadd>`

2.11 More APIs, SDKs, and Drivers for Extended Development Options

FairCom RTG V3 includes more SDKs and drivers in the development package. Although not required for integration with COBOL or Btrieve applications, they allow you to access your application's live data from other applications developed within supported languages and frameworks. Below is a list of APIs included with this release. The first part of each reference below is the directory name you will find in the *"drivers"* directory.

C

c.isam – ISAM API for C – Processes native binary C structures with exceptional control and speed

c.lowlevel – Low-Level API for C – Gives the programmer total control over indexes and data files for extreme performance and customizability but with potential incompatibility with other APIs and SQL

c.nav – c-treeDB API for C – The c-tree database API is easier to program than ISAM with slightly less control and speed

c.sql.direct – Direct SQL for C – Embeds SQL directly in C code

C++

cpp.nav – c-treeDB API for C++ – The object-oriented c-tree database API is easy to program and processes data quickly

cpp.replication – Replication API for C++ – Use the C++ Replication API to interface with Replication Manager to define, monitor and manage replication

C#

csharp.nav – c-treeDB API for C# – The object-oriented c-tree database API is easy to program and processes data quickly

csharp.sql.ado.net – SQL for ADO.NET and Tutorial in C# – Uses ADO.NET to work with the FairCom Database Engine like any other SQL database

csharp.sql.storedprocs – SQL Stored Procedure API for C# – Creates database stored procedures written in C#

Visual Basic

vb.nav – c-treeDB API for Visual Basic .NET – The object-oriented c-tree database API is easy to program and processes data quickly

C / C++ Callbacks

ctree.callbacks – Callback APIs for C and C++ – Allows C and C++ to create callbacks to process a wide-variety of database events

Java

java.jpa.nav – c-treeDB API for Java JPA– Uses Java, JPA, and the c-treeDB API for Java

java.nav – c-treeDB API for Java – The object-oriented c-tree database API is easy to program and processes data quickly

java.rest.replication – Replication API for Java is a tutorial in Java that uses the JSON RPC Replication API to replicate data between databases

java.sql.hibernate – SQL for Java Hibernate – Uses Hibernate and SQL to manage database records

java.sql.storedprocs – SQL Stored Procedure API for Java – Creates database stored procedures written in Java

JSON RPC

JSON RPC API is a Remote Procedure Control API that uses JSON over HTTP to allow all programming languages to remotely control the FairCom Database Engine for replication, SQL, and monitoring

MQTT

Message Queuing Telemetry Transport – MQTT is an industry-standard message queue protocol for sharing data and commands

MQTT Data Persistence API can create data persistence plans that automatically transform and persist JSON data sent to MQTT topics

Node.js

nodejs.nav – c-treeDB API for Node.js – Currently uses the C-style c-treeDB API

nodejs.rest.crud – REST API for Node.js – Allows Node.js to use the REST API to create databases, tables and indexes as well as query, retrieve, insert, update, and delete records

nodejs.sql – SQL for Node.js – Uses SQL to process relational data

Node-RED

node-red.rest.crud – REST API for Node-RED – Allows Node-RED to use the REST API to create databases, tables and indexes as well as query, retrieve, insert, update, and delete records

node-red.sql – SQL for nodeRED – Uses Node-RED and SQL to process database records

PHP

php.sql – SQL for PHP – Uses SQL to process relational data

php.sql.pdo – SQL for PHP PDO – Uses PDO to work with FairCom DB like any other SQL database

Python

python.nav – NAV API for Python – The NAV API is a new object-oriented navigational API that makes processing data very easy and fast

python.sql – SQL for Python – Uses SQL to process relational data

python.sql.sqlalchemy – SQLAlchemy for Python – Uses SQLAlchemy to work with FairCom DB like any other SQL database

REST

REST API makes it easy for any programming language to create databases, tables and indexes as well as query, retrieve, insert, update, and delete records

SQL

sql.cli – SQL for ISQL – Uses FairCom's Interactive SQL Utility (ISQL), which is a Command Line Interface (CLI)

sql.jdbc – JDBC – Provides a JDBC connection to Java programs and SQL tools to process relational data

sql.odbc – ODBC – Provides an ODBC connection to SQL tools and programs to process relational data

ThingWorx

thingworx.always-on – IoT Connector for ThingWorx – Uses FairCom's Always On Connector to ThingWorx to load data into ThingWorx

3. Go Bigger

The theme of FairCom RTG V3 and FairCom DB V12 is "Go BIG!"

We have increased the capacity of FairCom DB, which FairCom RTG is based on, 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.

3.1 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.

3.2 Millions of Open Files

In licensed enterprise editions of FairCom RTG V3 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*).

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.

3.3 128 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)

3.4 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.

3.5 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 41) 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 FairCom DB API level, the "scale" and "precision" properties have been adjusted to be consistent with the information in the system tables.

3.6 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.

4. Go Faster - By Simply Upgrading Your Server

No coding necessary...install the FairCom RTG Server (and, in some cases, add a few keywords) and your application goes faster.

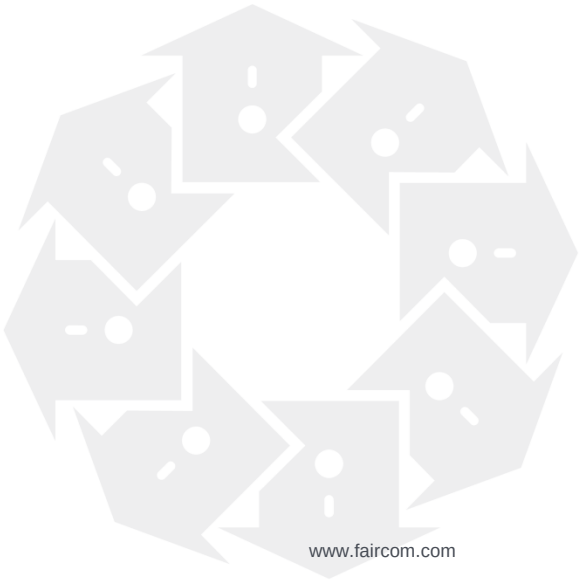
4.1 Up to 3x Faster Overall Performance

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.

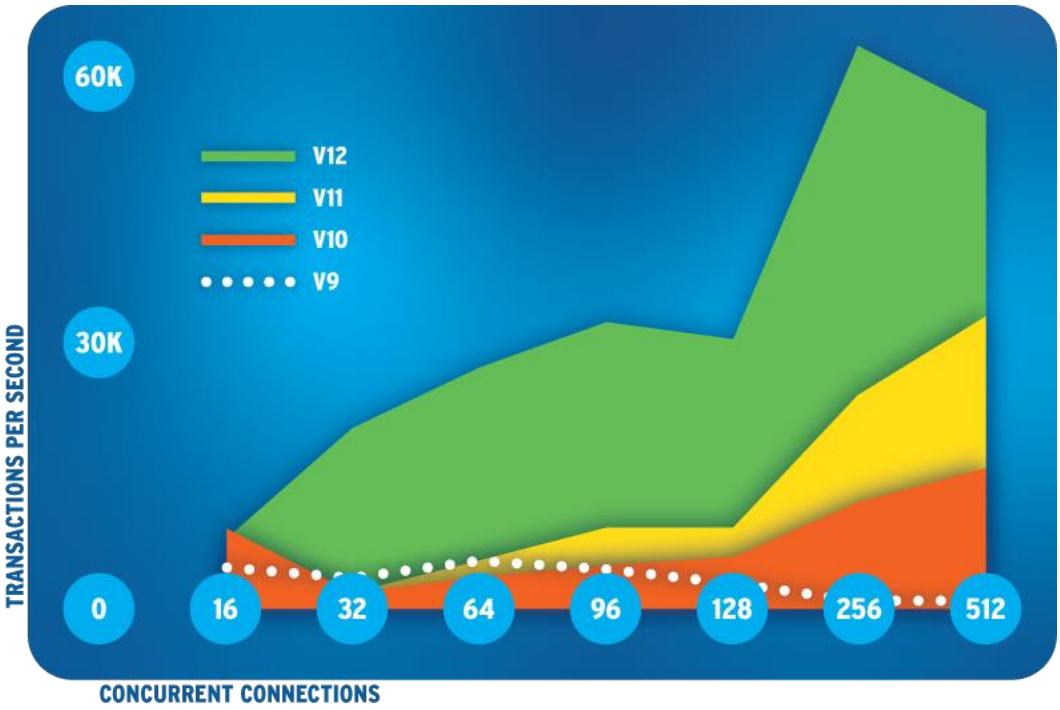
Overall performance gains of FairCom DB and FairCom RTG are again included in this release. Multiple areas from file open and close operations to index key reads have been profiled for capacity limitations. This release removes many bottlenecks for additional transaction throughput.

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.



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

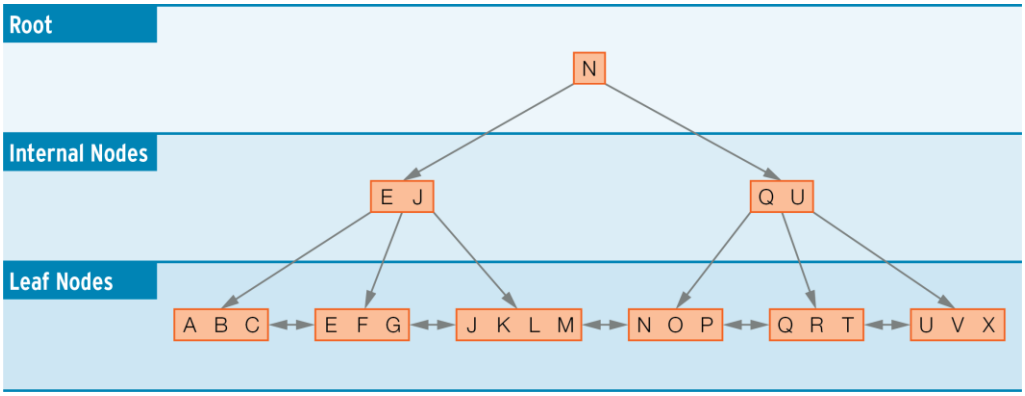
FairCom DB indexing has always been optimized for the fastest possible data access. Indexes are composed of keys. However, not all keys are composed the same. For example, consider indexing based on file paths. Path strings are highly variable in length. Indexing data based on this type of key definition 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.

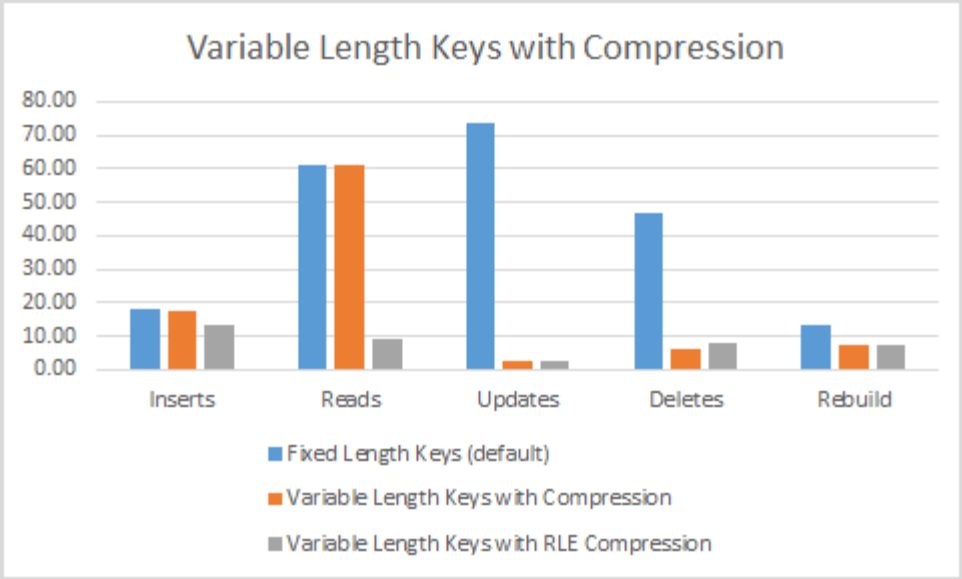
By default, FairCom DB indexes store keys as fixed-length values. This requires few CPU cycles however, increases storage space and resulting I/O. Taking advantage of the fact CPU processing is orders of magnitude faster than I/O, we created a new compressed, variable-length index structure significantly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

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*).

Key Compression with FairCom RTG

The `<keycompress>` option defaults to index compression.

The following `<keycompress>` suboptions specify which compression type to use:

- `<rle>` - Indicates to compress the whole key using a simple RLE algorithm.
- `<padding>` - Indicates to compress the padding characters of the key.

The new index compression does not support the `<leading>` compression type. The old index compression can be used by disabling `<keycompress vlennod>` (enabled by default), which allows the `<leading>` and `<padding>` to be specified. The new `<rle>` option cannot be specified when the `<keycompress vlennod>` attribute is disabled. The `<rle>` and `<padding>` sub-options are mutually exclusive unlike the old index compression where it was possible to combine `<leading>` and `<padding>`.

The following examples turn on the new index compression:

```
<keycompress/>
<keycompress>1</keycompress>
<keycompress><rle/></keycompress>
<keycompression><padding>1</padding></keycompression>
```

The following examples enabled the old index compression:

```
<keycompress vlennod="no"/>
<keycompress vlennod="0">1</keycompress>
<keycompress vlennod="false"><leading>1</leading></keycompress>
<keycompression vlennod="n"><leading/><padding/></keycompression>
```

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,

};
```

4.3 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.

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:

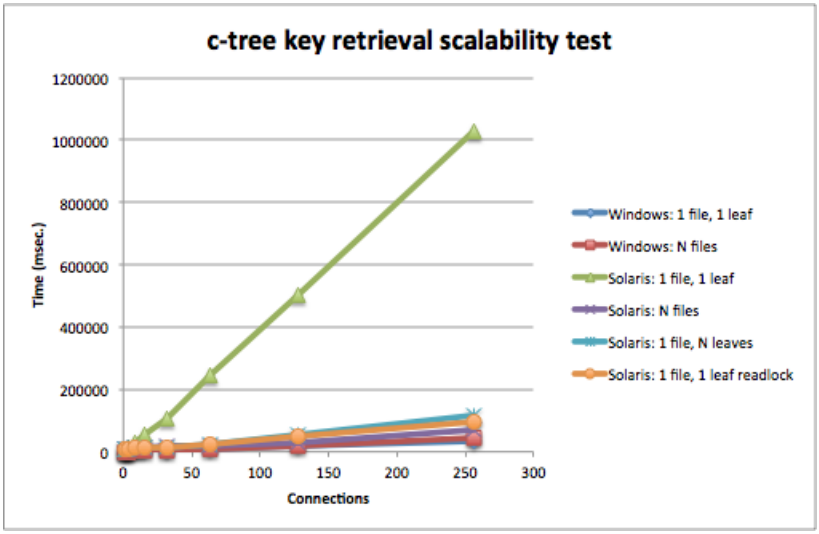
1. An index that uses key compression.
2. An index that is under transaction control.

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.



Extend leaf node read lock optimization to moving right or left at leaf level

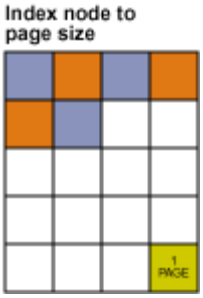
When key lookup operations move right or left at the leaf level, the call to search the index cache for an index buffer that holds the specified node was not taking advantage of the leaf node read lock optimization. The logic has been modified to do this. This change could boost performance of key lookups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

4.4 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. (This is configurable with the *sect* parameter of standalone applications; each *sect* = 128 bytes.)

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

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`.

Compatibility

The page (node) size is a permanent attribute of an index when it is created. Index nodes remains that size until rebuilt or compacted with a different size. There are limitations with existing files using a new larger page size.

- c-tree can open existing indexes with a *smaller* page size than currently configured. There is a minor memory use trade off in doing so. As each index node maps to a single server cache page, and the server cache page is allocated as a page size, the unused space in the cache page is lost. This can be significant. If a server configured for a 32K page size index cache is at 100% capacity of 8K index nodes, *up to 75% of cache memory is unused.* For very large caches this is significant.

  It will be extremely beneficial to rebuild existing indexes and take full advantage of both increased index page size benefits as well as full cache memory usage.

  Opening indexes created with page sizes larger than currently configured results in error 40 (**KSIZ_ERR**).
- *Superfiles can ONLY be opened with the same configured page size as they were originally created.* This has important implications with critical c-tree housekeeping files:
  - *FAIRCOM.FCS* - This file maintains all user, group and password hash information.
  - *ctdbdict.fsd* - This file maintains the catalog of available databases (SQL and c-treeDB).
  - *<database_name>.fdd* - This file maintains the SQL database system tables (catalog or dictionary).

  Using these existing superfiles with a different page size will result in a server startup failure. Opening a superfile with a different page size also results in error 40 (**KSIZ_ERR)**. You will find this message logged in *CTSTATUS.FCS* should the server make this failed attempt.

Rebuilding Existing Indexes

The following options are available for rebuilding indexes to take advantage of increased page size.

ALWAYS MAKE CLEAN BACKUP COPIES OF THESE FILES BEFORE YOU BEGIN THESE PROCEDURES

The index rebuild switch *-rebuild* for the **ctutil** utility is the quickest, easiest option for most indexes. Simply be sure you have set the new `PAGE_SIZE` setting in *ctsrvr.cfg* and then perform a rebuild using **ctutil** *-rebuild* pass the new *sect* size.

```
ctutil -rebuild myfile
```

Remember not to include the file extension on your file name.

For superfiles, use the superfile compact utility, **ctscmp**. This utility has an option for passing the sect size. *Remember, each sect = 128 bytes. For 32768 your sect size will be 256, as shown in the following example:*

```
> ctscmp ctdbdict.fsd Y 256
```

4.5 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

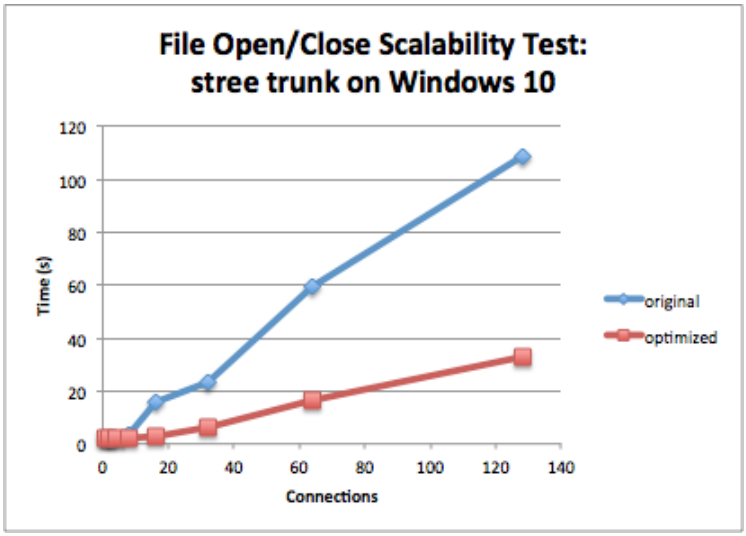
Files	Windows 10	Solaris 11	Centos6
64	60.170	50.911	11.589
128	113.519	78.803	21.287

With file open/close optimizations:

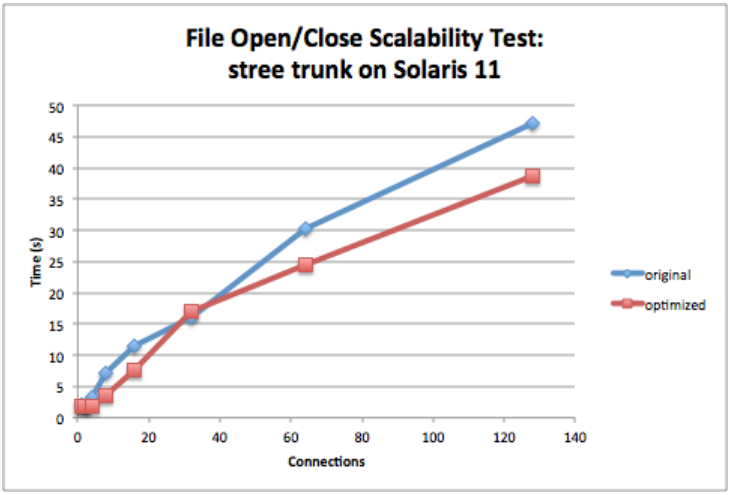
Files	Windows 10	Solaris 11	Centos6
1	1.804	1.894	2.713
2	1.795	1.771	2.687
4	1.940	2.149	2.597
8	2.036	2.706	2.575
16	2.357	3.686	2.492
32	5.241	7.010	5.355
64	12.379	13.462	10.452
128	25.849	27.322	18.778

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

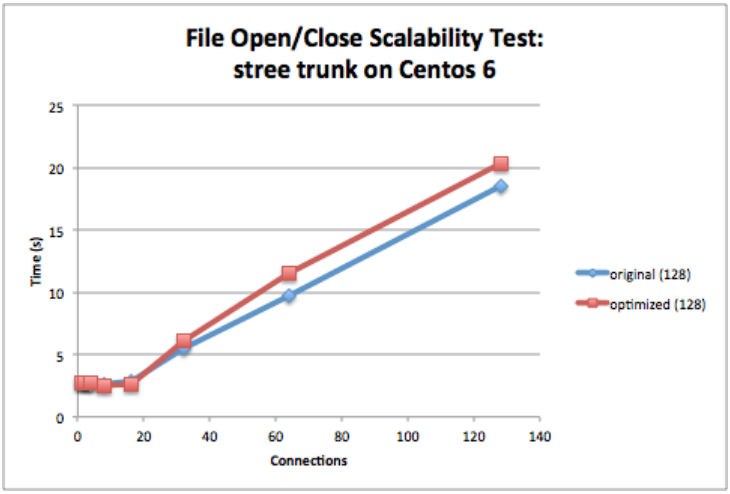
Windows 10: **77% reduction** in time (from 113.5 sec. to 25.8 sec.)



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.

4.6 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.

4.7 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.

4.8 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
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
```

4.9 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.

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 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.

4.10 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.

4.11 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.

5. SQL

Relational SQL access remains a powerful force for data reporting and analytics. FairCom RTG allows legacy data to be linked to SQL. FairCom RTG V3 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.

5.1 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*).

5.2 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
```

5.3 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.

5.4 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
```

5.5 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));
```

5.6 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
```

5.7 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 FairCom DB API 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.

5.8 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.

5.9 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.

5.10 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"}
-------------------------------------------------
```

5.11 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.

5.12 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
```

5.13 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.

5.14 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*.

6. 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.

6.1 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.

6.2 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.

6.3 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
```

6.4 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.

6.5 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.

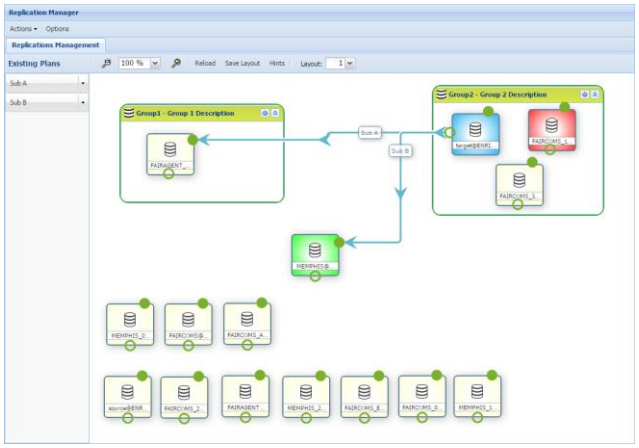
Slow performance was sometimes experienced when using the **ctrdmp** (https://docs.faircom.com/doc**/ctserver/ctrdmp-util.htm**) utility to restore a large number of files and transaction activity occurred during the dynamic dump. Recovery utility performance has been improved for these activities. This 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.

7. Data Replication

With this release, FairCom announces the next phase of FairCom replication. This new advancement brings new capabilities to tackle the most demanding of replication requirements.

7.1 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/docs/en/UUID-d8467037-78cc-2e04-2dc7-5293fd5562af.html#UUID-d8467037-78cc-2e04-2dc7-5293fd5562af*)
- *Replication Extensions* (https://docs.faircom.com/doc/*replication_extension/*)

7.2 High Availability

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. Configuration

This chapter provides a single location for finding all of the configuration options new with V3. Some items may be duplicated in other parts of this guide. This chapter includes configuration options for FairCom DB as well as FairCom RTG.

8.1 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.

8.2 FairCom RTG TLS (SSL) for Encrypted Network Communications Support

This modification introduces two new attributes to the `<instance>` configuration element to support TLS (SSL) communication in FairCom RTG:

The `<instance ssl="">` attribute specifies if a secure connection should be used to connect to the FairCom RTG server. It accepts "`yes`" or "`no`" as possible values and defaults to "`no`".

The `<instance sslcert="">` attribute specifies the file containing a specific public certificate of the FairCom RTG Server.

8.3 <instance endiancheck> Keyword to Relax Server's Endianness Check

The FairCom RTG data files do not have file schema ("DODA" in c-tree terminology), therefore FairCom RTG cannot take advantage of FairCom's Netformat communication logic, which automatically flips bytes between dissimilar platforms. This logic checks at connection time to ensure that the byte endianness of both client and server is the same (if it is not, the connection is terminated and the operation fails). In cases where this logic is not necessary (e.g., files without numeric data types), the byte endianess check can be relaxed so the connection can succeed.

The configuration attribute `<instance endiancheck="no">` allows skipping the byte endianess check. This attribute defaults to "yes."

8.4 <instance ctshmemdir> to Set Shared Memory Directory under Unix

On Unix platforms, the FairCom RTG server shares a directory with the FairCom RTG clients when they communicate via the shared memory protocol. It is possible to configure the FairCom RTG server to change the default shared memory directory with configuration keyword `SHMEM_DIRECTORY`. Using different shared memory directories allows running multiple FairCom RTG servers with the same name on the same machine. To configure the FairCom RTG clients to set the shared memory directory, it is necessary to set the c-tree global variable `ctshmemdir`.

This revision implements a new `<instance ctshmemdir>` attribute to configure the shared memory directory for the `<instance>` by setting the c-tree global variable `ctshmemdir`.

The `<instance ctshmemdir>` is ignored under Windows as it is meaningful only for Unix platforms.

8.5 <redirinstance> Support to Fallback to Default File System

This modification introduces support to configure `<redirinstance>` so that the default file system is used to handle files. This functionality is available only for the ExtFH interface (because only Micro Focus COBOL supports the fallback to the default file system when the `DYNREDIR=` environment variable is in use).

The `<redirinstance>` can be configured without setting the `<redirinstance lib>` attribute if the runtime supports fallback to the default file system.

If the `<redirinstance>` is configured without the `<redirinstance lib>` attribute and the runtime does *not* support fallback to the default file system, a warning message "Invalid <redirinstance> configuration" is logged and the operation fails with an error.

8.6 <forcedelete> Configuration Option to Force Deletion of Orphan Files

The COBOL operation DELETE FILE fails with COBOL error **98** (corrupted file) on orphan files that are data files that are missing their relative index file. The DELETE FILE operation on Micro Focus instead, deletes the orphan files and returns successful.

The new configuration option `<forcedelete>` forces DELETE FILE operations to delete any orphan file left when set to "yes". When it is set to "no", it causes DELETE FILE to return an error and leave the orphan file on disk.

By default, `<forcedelete>` is set to "no" except for ExtFH drivers where it is set to "yes" to match Micro Focus COBOL behavior.

Before the introduction of <forcedelete>, the default behavior for FairCom RTG ExtFh driver was to fail DELETE FILE on orphan data files (missing its index file) and succeed on orphan index files (missing its data file) to match the behavior of old Micro focus COBOL products. More recent Micro Focus product behavior is to remove any orphan file and return success.

8.7 <rowid> and <rowid size> Attributes

The "size" attribute for the `<rowid>` tag can be used to set the size of the *rowid* hidden field in the record header. The default value is 8 for COBOL (4 for BTRV). Accepted values are 8 and 4.

```
<rowid size="8">yes</rowid>
```

Compatibility: The size setting of `<rowid>` is taken into consideration even if `<rowid>` is set to `no`, affecting other FairCom RTG features. Setting it to `8` (for COBOL, not setting a value has the same effect) makes new files incompatible with previous FairCom RTG versions.

Therefore, to be sure that files created with V3 are compatible with older versions, it is necessary to include the following in *ctree.conf*:

```
<rowid size="4">no</rowid>
```

The `<filecopy>` overwrite attribute allows existing files to be overwritten.

8.8 <filepool> Size and <inpool> Options

The `<filepool>` configuration keyword has been enhanced with the `<filepool>` *size=* attribute, which optionally sets the maximum number of files to keep in the pool. As a refresher, recall that the `<inpool>` configuration keyword defines if a file can be included in the file pool.

8.9 <keycompress> Option to Create Files with Key Compression

The <keycompress> option now defaults to the new index compression. In internal testing, FairCom has observed a 6X reduction in index sizes, which led to a 2X increase in transaction throughput.

The following <keycompress> sub-options specify which compression type to use:

- <rle> - Indicate to compress the whole key using a simple RLE algorithm.
- <padding> - Indicate to compress the padding characters of the key.

The new index compression does not support the <leading> compression type.

The <rle> and <padding> sub-options are mutually exclusive unlike the old index compression where it was possible to combine <leading> and <padding>. The new index compression does not support <leading> compression type.

The old index compression can still be used to create files by disabling the <keycompress vlennod> attribute which is enabled by default. When <keycompress vlennod> attribute is disabled, the old index compression sub-options <leading> and <padding> can be specified. Of course the new <rle> cannot be specified when the <keycompress vlennod> attribute is disabled.

The following example turn on the new index compression:

```
<keycompress/>
<keycompress>1</keycompress>
<keycompress><rle/></keycompress>
<keycompression><padding>1</padding></keycompression>
```

The following examples enabled the old index compression:

```
<keycompress vlennod="no"/>
<keycompress vlennod="0">1</keycompress>
<keycompress vlennod="false"><leading>1</leading></keycompress>
<keycompression vlennod="n"><leading/><padding/></keycompression>
```

8.10 <scancache> Scanning Caching Strategy Option

The FairCom RTG <scancache> configuration option allows setting the cache strategy to be used for reading one file or a set of files. The <scancache> option is a boolean that is enabled with <scancache>yes</scancache>, or disabled with <scancache>no</scancache>.

If <scancache>no</scancache> is used, the data cache buffer pages are managed using c-tree's default reuse scheme, which reassigns a cache page anytime a read requires a page that

is not already in cache. This strategy can allow a single user doing many reads from a single file to take over a large percentage of the total cache for that file.

If this option is enabled with `<scancache>yes</scancache>`, all the standard cache routines are used unless the read requires a cache page that is not already in cache. In that case, it limits each user to a maximum of 2 cache pages. The significance of this strategy is that a large set of data record read operations can be performed using only two cache pages with little or no decrease in performance compared to no restrictions on cache usage. Moreover, such restricted use may help other aspects of system performance since the bulk of data cache can be used for other files and/or other users. A large traversal of the data file by a single user will not replace other cache pages that may be useful to other users.

For more about this caching technique, see *Scanner Cache /doc/ctreeplus/53846.htm* in the FairCom DB Developer's Guide.

8.11 <memoryfile persist> Attribute to Specify if Memory File Is Removed at Disconnection

Once created, memory files are available until they are either removed or until the FairCom RTG server is shut down. This allows the memory file to be available for multiple sessions to multiple applications while the FairCom RTG server is active.

Typically the deletion of the memory file is performed by the application that created it as it has knowledge of its existence. However, if the application disconnects from the FairCom RTG server in a non-controlled way (i.e., crash), the memory file remains available to other applications which might not know of its existence and therefore will not be able to delete it. Overall, this may cause an accumulation of orphan memory files and an exhaustion of memory.

The `<memoryfile persist>` configuration attribute specifies if the memory file being created will be available while the FairCom RTG server is active or if it is automatically removed when the application that created it disconnects from the FairCom RTG server (either in a controlled way or as a result of a crash).

`<memoryfile persist="yes">` - Once created, the memory file is available until the FairCom RTG server is shut down or until any application manually deletes it.

`<memoryfile persist="no">` - The memory file is available until the application that created it disconnects from the FairCom RTG server.

8.12 <prefetch ttl> Attribute to Define How Long a Set of Prefetched Records Remains Valid

FairCom RTG has the ability to "prefetch" records to improve efficiency. When a client requests a record from the server, it can be more efficient to fetch several records at a time because they can share the same communications overhead. If the client needs another record, it may find that record has already been prefetched, making it available with no additional overhead.

In some situations, the prefetched records have been waiting long enough that they are no longer valid. This is particularly true if the application has been waiting for user input and other users have attempted to access the same records. In this case, it is necessary to invalidate the prefectched records so they will be fetched again when needed.

This modification introduces a new `<prefetch ttl>` attribute (`ttl` is "time-to-live") that defines the number of seconds a set of prefetched records remains valid. By default, the value is set to 0 seconds which means that prefetched records do not expire. When `<prefetch>` is enabled and `<prefetch ttl>` attribute is set, the records prefetched by a sequential read that are not consumed within the number of seconds specified in the `<prefetch ttl>` attribute will be discarded and the next consecutive sequential operation will cause a new prefetch to be requested from the server.

Example:

The following example enables `<prefetch>` reading for at least 5 records allowing 2 seconds to consume the prefetched records:

```
<config>
   <instance server="FAIRCOMS">
      <file>
         <prefetch records="5" ttl="2"/>
      </file>
   </instance>
</config>
```

8.13 <localinstance>

The <localinstance> element specifies instance-wide configurations for the standalone driver.

Note: **This element is only used for standalone versions of the product.**

Attributes

Attribute	Description	Default value
bufs	Specifies the number of index file cache buffers. Minimum of 3 required. Maximum is 32,767. Increasing these values can improve performance during bulk data loading.	1280
dbufs	Specifies the number of data file cache buffers. Minimum of 3 required. Maximum is 32,767. Increasing these values can improve performance during bulk data loading.	1280
fils	Specifies the initial block of file structures to allocate. Whenever the number of files required exceeds this initial amount, another block of file structures using this number is automatically allocated. Each index, whether it is a member of an index file or in a file by itself counts toward this parameter.	32
sect	Specifies the number of node sectors. Minimum of 1 required. This parameter multiplied by 128 equals the index node size. (The default of 256 = 32,768 bytes.)	256
logpath	Specifies the file path for the transaction processing log files.	""
temppath	Specifies the file path for storing temporary files.	""
sortmem	Specifies the size of sort buffers used by the local database instance.	100 MB

Example

```
<localinstance bufs="1000" fils="64" sect="128" dbufs="1000" logpath="/data/logs">
    ...
</localinstance>
```

Requirement

`<localinstance>` support requires the presence of a FairCom Standalone DLL or .SO, named *ctreestd.dll* or *ctreestd.so* for Linux/Unix systems. Contact FairCom for availability options.

Limitations

- Standalone usage does not support transactions. In essence, any update performed on a file defined under `<localinstance>` is committed immediately and cannot be rolled back. A "ROLLBACK TRANSACTION" operation is ignored and a warning message is logged.
- Standalone usage does not support `<runitlockdetect>`.
- Standalone usage does not support turning off `<optimisticadd>`, i.e., `<optimisticadd>no</optimisticadd>`

8.14 <startonread> Option to Improve Performance of START and READ NEXT/PREVIOUS Operations

The `<startonread>` configuration option improves performance of COBOL applications that perform a large number of START operations all followed by READ sequential operations, such as READ NEXT or READ PREVIOUS.

When `<startonread>` is enabled, FairCom RTG does not execute the START operations immediately. Instead, it waits for the subsequent READ sequential operation (READ NEXT or READ PREVIOUS). The effect is that START operations return immediately but never fail even if the specified key is not valid. It is the subsequent READ sequential operation that executes the START operation and eventually returns the error. For this reason, we do not recommend using this option unless the user understands the risks.

If the FairCom RTG server does not support `<startonread>`, the FairCom RTG client fails and logs error message "server does not support <startonread> configuration."

SYMPTOM: Performance of START + READ NEXT / PREVIOUS is worse when <prefetch> is enabled and gets worse increasing the number of prefetched records with <prefetch records> configuration attribute.

PROBLEM: The RTG prefetch logic gets triggered by the second consecutive READ NEXT or READ PREVIOUS operation as a consequence, a loop of START operations followed by READ NEXT operations will never trigger the prefetch logic. However the logic that sends the request to the RTG server to read the next record, prepares the RPC network buffer to receive the prefetched records when just the <prefetch> configuration option is enabled. The result is that an useless memory allocation is performed during a READ NEXT operation that does not prefetch records. This unused amount of memory can be significant for large <prefetch records> attribute settings. We noticed that large memory allocations are sometime costly in terms of performance depending of the platform.

SOLUTION: Change the logic that calculates the RPC network output buffer to include the prefetch buffer size (CT_FILE->fetsiz) only if a prefetch request is actually sent to the server.

8.15 New <log> <debug> Attributes

The following new log debug attributes have been implemented:

<lock> Attribute for <log><debug>

This attribute enables logging of diagnostics messages about record/file locking. When enabled, it writes a DEBUG LOCK message in the log each time a lock request is performed.

<switcher> Logging Option

This option logs all calls to the FairCom RTG switching logic, which is active when switching between c-tree and the native COBOL file handler.

<startonread> Logging Option

This option logs critical <startonread> events:

- With <startonread> enabled, START always returns success; the next READ sequential performs the START and returns an error if START fails.

When enabled, this option logs: "previous START returned success due to <startonread> but should have returned error".

Full details on the log debug is available in te FairCom RTG guide: *<debug>* (https://docs.faircom.com/doc/*ctcobol/_debug_.htm*)

8.16 New <log> <error> Attributes

The following new elements have been added to the `<log> <error>` support:

Name	Description	RTG Error	Accepted Values
<locked>	This value masks record-locked errors. *See Note below.	5	yes and no
<missingfile>	This value masks file-not-found errors. *See Note below.	15	yes and no
<undefined>	This value masks no-current-record-position errors. *See Note below.	9	yes and no

Note:

These masks can be useful if you encounter a large number of error **9:0** (undefined record position) in your FairCom RTG logs and need a way to mask them.

In the FairCom RTG log file, these errors can be identified by the first of the three colon-separated error values:

```
RTG Error : Underlying c-tree API Error : Operating System Error
```

For example, a record locked error will display the following (notice 42 is the c-tree error for record locked):

```
ERROR 5:42:0
```

Examples

The following example enables the logging of errors only:

```
<log file=mylog.txt>
    <error>yes</error>
</log>
```

The following example enables the logging of end-of-file errors but not of record-not-found errors:

```
<log file=mylog.txt>
    <error>
            <atend>yes</atend>
            <notfound>no</notfound>
    </error>
</log>
```

Full details on log error attributes are available in the FairCom RTG guide: *<error>* (https://docs.faircom.com/doc/*ctcobol/_error_.htm*)

8.17 FairCom DB Configuration Options

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 increased 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)

SQL

VSS_WRITER_QUIESCE_TIMEOUT (https://docs.faircom.com/doc/ctserver/vss-writer-quiesce-timeout-config.htm)
allows setting a default timeout value

SQL_SERVER_LOG_SIZE (https://docs.faircom.com/doc/sqlops/SQL_SERVER_LOG_SIZE.htm) limits setting error 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

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

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

9. FairCom RTG Security

FairCom RTG supports robust security options. This chapter explaimns those options and lists the latest security updates.

For information about secure optins for utilities, see *Security (page 96) in the FairCom RTG Utilities chapter.*

9.1 Secure SSL Communication

FairCom RTG Supports TLS/SSL Protecting Data in Transit for Network Communication

FairCom RTG applications can secure data in transit between c-tree network clients and FairCom DB Servers. Transport Layer Security (TLS, also commonly referred to as its predecessor SSL, Secure Sockets Layer) is a cryptographic protocol designed for secure network communications using public key cryptography for authentication of the communicating party. Symmetric cryptography is used to encrypt transmitted data over the wire. FairCom DB TLS relies on OpenSSL toolkit support (version 1.0.2g) and implements TLS protocol V1.2 exclusively, as earlier versions of TLS (and predecessor SSL) protocols contain known and exploited vulnerabilities. FairCom DB supports TLS via TCP/IP communications protocols (IPv4 and IPv6). (For more about TLS, see SSL/TLS in Wikipedia (https://en.wikipedia.org/wiki/Transport_Layer_Security).)

Two modes of TLS connections are available: basic and peer authenticated. Basic TLS connections are encrypted using only a server-side certificate; there is no local certificate requirement for a client. This makes deployment and management of secured connections easy.

TLS Certificates

It is the server administrator's responsibility to ensure a correct and valid certificate pair, as well as proper configuration of allowed TLS connections.

Creation and management of TLS certificates, as well as use of a Certification Authority (CA), is beyond the scope of this document. Consult OpenSSL and other TLS supporting documentation and be sure you firmly grasp all details regarding use of TLS for network security before deploying.

Server certificates may be created and provided as two separate files: a certificate and a key. They can also be combined into a single file. There is no required file naming convention. Certificate files are usually created and/or provided as Base64 encoded X.509 certificate files and denoted with a *.pem* extension (Privacy Enhanced Mail). They can be identified with this set of surrounding identifiers within the file:

```
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
```

The private key is likewise identified:

```
-----BEGIN PRIVATE KEY-----
-----END PRIVATE KEY-----
```

The private key is often included in the same certificate file or as a separate *.key* file.

A certificate key can be optionally passphrase protected. However, this then requires the passphrase to be presented at server startup. FairCom DB key store files provide this ability.

Always securely maintain key files. Store key files only in permissions protected server areas and never distribute.

Peer Authentication - TLS Connection with Server Certificate Validation

By default, a c-tree client requires a PEM file containing the server public certificate—**ONLY the public certificate, not the server private key**.

Important: The server private key should be securely maintained only at the FairCom Server location at all times.

By default, c-tree client libraries use the file *ctsrvr.pem* in the client process' working directory when connecting.

When the client does not use a server certificate, the connection is encrypted, but there is no guarantee that the client is connected to that specific server. This implies that a "man in the middle" attack could be possible.

Server-Side Configuration

To enable TLS (SSL), add a `SUBSYSTEM COMM_PROTOCOL SSL` section to *ctsrvr.cfg* containing your specified TLS configuration options. Supported options include:

- `SERVER_CERTIFICATE_FILE` (required) - This is the name of a PEM-encoded certificate file containing FairCom DB Server's certificate. It can also optionally include the private key. The server certificate can be self-signed.
- `SERVER_PRIVATE_KEY_FILE` (optional) - If the private key is in a different file than the certificate, use this option to indicate the name of the file containing the private key. If this option is not specified, the private key is expected to be found in the file whose name is specified by the `SERVER_CERTIFICATE_FILE` option.
- `SERVER_ENCRYPTED_STORE_FILE` (optional) - If the private key is encrypted, use the **ctcpvf** utility to create an encrypted store file containing the passphrase used to decrypt the private key and specify the name of the encrypted store file with this option.
- `SSL_CONNECTIONS_ONLY` (optional) - Default: NO. If this option is specified with a value of YES, only TCP/IP connections that use SSL are permitted to connect to FairCom DB Server. Otherwise, both unencrypted and SSL-enabled TCP/IP connections are allowed.
- `SSL_CIPHERS` (optional) - Default: `AES256-SHA256:AES256-GCM-SHA38:DHE-RSA-AES256-SHA256`
  If this option is specified, it sets the encryption ciphers that are allowed to be used for encrypting the SSL connection. The default specifies full AES 256-bit encryption. Ciphers are

separated by a colon, (":"). An exclamation point ("!") symbol explicitly disables a cipher. `@STRENGTH` sorts the list in order of encryption algorithm key length. For more information, see https://www.openssl.org/ (https://www.openssl.org/).

Standard c-tree TCP/IP ports are used for connections regardless of TLS configuration. That is, a single ISAM port will handle both TLS encrypted and non-encrypted connections, and likewise for the SQL port. There is no need for separate port configurations.

Client-Side Configuration

TLS (SSL) communication support is enabled in FairCom RTG through two new attributes to the `<instance>` configuration element.

- `<instance ssl="">` specifies if a secure connection should be used to connect to FairCom RTG server. It accepts "yes" or "no" as possible values and defaults to "no".
- `<instance sslcert="">` specifies a server public certificate when server certificate validation enabled.

Example

```
<config>
       <instance  ssl="yes"  sslcert="ctsrvr.pem"  server="FAIRCOMS@localhost"   user="ADMIN"
password="ADMIN"  connect="no"  versioncheck="no">
...
      </instance>
</config>
```

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.

9.2 FairCom RTG Now Supports c-tree File Ownership Attributes

FairCom RTG has extensive file ownership and access controls. Administrators can assign read, write, delete, and definition change access controls per file. In addition, c-tree supports inheritable user, group and world access permissions (similar to Unix filesystem permissions.) By combining operating system permissions, c-tree access permissions, and file access rights, FairCom RTG becomes a highly secure data access layer.

Previously, FairCom RTG-created files lacked ownership attributes. Now, FairCom RTG can be configured, per file, with an explicit file ownership allowing for advanced administrative file access controls. File ownership is displayed by administrator tools such as **ctadmn**. For backward compatibility file permissions are set to allow access to everybody.

FairCom RTG File Permissions Support

The <permission> configuration option allows you to set file read/write access, file redefinition, and file delete permissions for the file owner, group members, and all other users.

The <permission> configuration option can have up to 3 possible options:

- <owner> - set permissions for the file owner
- <group> - set permissions for the group members of file owner
- <world> - set permissions for the rest of all users

In turn <owner>, <group>, and <world> can each have up to 4 possible options:

- <read> - set file access permission
- <write> - set record write permission
- <def> - set file redefinition permission
- <delete> - set file delete permission

If the <permission> keyword is *not* specified in the configuration file, the default behavior is to allow all permissions to all users. If <permission> *is* specified in the configuration file, the default behavior is to deny permission unless it is specifically specified with configuration options.

Example 1

The following permission configuration permits file deletion and file redefinition only to the file owner while all users can access the file for reading but only members of the file owner group are allowed to write records:

```
<file dir="protected">
 <permission>
  <owner><read>yes</read><write>yes</write><def>yes<def/><delete>yes</delete></owner>
  <group><read>yes</read><write>yes</write><def>no<def/><delete>no</delete></group>
  <world><read>yes</read><write>no</write><def>no<def/><delete>no</delete></world>
 </permission>
</file>
```

Example 2

The following permission configuration permits file deletion and file redefinition only to the file owner while all users can access the file for reading but only members of the file owner group are allowed to write records:

```
<file dir="protected">
 <permission>
  <owner><read/><write/><def/><delete/></owner>
  <group><read/><write/></group>
  <world><read/></world>
 </permission>
</file>
```

9.3 cmdset Support Added to FairCom RTG

FairCom DB V12 and FairCom RTG V3 now offer improved security around password handling. This modification introduces support for authorization files in FairCom DB command-line utilities. This feature enhances security because it eliminates the need to send a visible password from the command line. The authorization file is created with the c-tree **ctcmdset** utility.

To use this utility, first create a plain text password file similar to the following (note for our example, we called this file *password.cfg*):

```
; User Id
USERID ADMIN
; User Password
PASSWD <pass>
```

Then execute **ctcmdset** to create a masked version of the *password.cfg* file, as shown:

```
./ctcmdset password.cfg
```

The output from the **ctcmdset** utility will be a file with a *.set* file extension, which would be *password.set* in our example.

To utilize this new masked password file with the various command line utilities, use a *-1* switch:

```
ctadmn -1 password.set
```

When using this *.set* file and the *-1* switch, you won't need to provide the user ID or password to the various FairCom DB command-line utilities. The following utilities support this command-line parameter to prevent the password from being seen in the clear:

- ctadmn
- ctalog
- ctdump
- ctfilblkif
- ctflush
- ctgendef
- ctinfo
- ctpass
- ctquiet
- ctreplagent
- ctsmon
- ctstat
- ctstop
- ctsysm
- ctutil
- dfkctl
- repadm
- sa_admin

cmdset Support for the <instance> Element

The FairCom RTG `<instance>` element in *ctree.conf* specifies instance-wide configurations. Each instance represents a connection to the FairCom RTG server. The `<instance>` element supports an *authfile* attribute to specify an encrypted authorization file created with the **ctcmdset** utility /doc/cmdline/ctcmdset-util.htm. If both *authfile* and *user* are specified, *authfile* takes precedence.

9.4 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.

9.5 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.

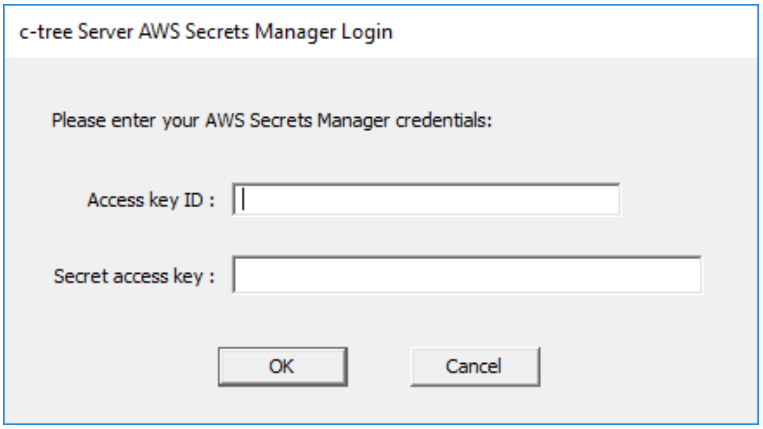
Note: When using AWS key store support on servers that are licensed for multiple remote desktop sessions, the password prompt may not be presented unless the database process is run as a normal process (not a Windows service). Because this might involve file permissions changes, it is recommended to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

9.6 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)
```

9.7 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.

9.8 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.

9.9 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 49).*

9.10 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.

9.11 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.

9.12 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]
```

9.13 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).**

10. Goal: Zero Administration

FairCom RTG strives to be "DBA Free" for easy deployment and low TCO (total cost of ownership).

10.1 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.

10.2 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.

10.3 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() in the programming guide for details.**

10.4 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.

10.5 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.

11. Utilities

Many existing utilities have been updated. This section describes scriptable command utilities. Be sure to check out new browser-based tools as well!

11.1 ctutil

ctutil Changes

The following changes have been made to the **ctutil** utility:

1. A new *-f* option for **ctutil** *-info* and *-unload* commands (**ctutil** *-info -f* and *-unload -f*) to open a file even if it is corrupted (i.e., file open returns c-tree error 14 **FCRP_ERR**). Note this modification deprecates the generic **ctutil** option *-s* that was doing the same thing and was used only by *-info* and *-unload* commands.
2. The **ctutil** *-load* command now supports a "new file" option: *-n*. When this option is used, any data in the destination file is eliminated before the records are loaded from the source file.
3. **ctutil** *-tron file T|P|F|W* - now displays a message at the conclusion of the requested operation indicating success or an error message in the case of a failed operation. As a refresher, recall you don't include the file extension when passing the file name (e.g. *myfile* instead of the full file name with extension, like *myfile.dat*). The "*T|P|F|W*" settings are:

   T – enable transaction support with full logging (Transaction (TRNLOG) mode);

   P – enable transaction support without logging (Pre Image (PREIMG) mode);

   F – disable transaction support – indicating your data integrity will be impacted;

   W – enable synchronous writes to disk – this mode is slightly faster than *T*, but provides better data integrity than mode *F*.
4. **ctutil** *-fileid* resets the unique file ID of a file that has been copied outside of FairCom RTG (for example a file system **copy** command).
5. **ctutil** *-compress/-uncompress* – now reports a message at the conclusion of the operation indicating if the file was successfully compressed or uncompressed.
6. **ctutil** *-sqlrefresh* is very similar to "-sqlize", "-sqlrefresh":
   a. it can be used only if the table was already sqlized
   b. it refreshes the table structure maintaining the existing grants and synonyms
   c. does not support "public" option available in "-sqlize" since authorizations are kept from the original sqlize and grants that occurred in SQL
7. **ctutil** *-run* executes multiple **ctutil** commands in a single run to avoid program initialization overhead, see **ctutil** *-run (https://docs.faircom.com/doc/ctcobol/-run.htm)*
8. The following obsolete **ctutil** commands have been deprecated:
   - *-param* (display parameter file) is obsolete.
   - *-loadtext* (import from line sequential file) has been replaced by the *-t* option of the *-load* command.
   - *-unloadtext* (export from line sequential file) has been replaced by the *-t* option of the *-unload* command.

ctutil -load Option '-n' to Empty Destination File Before Loading Records

The **ctutil** *-load* command now supports a "new file" option: *-n*. When this option is used, any data in the destination file is eliminated before the records are loaded from the source file.

ctutil -tron Updated

The **ctutil** *-tron* command has been updated so that a successful **ctutil** *-tron* command displays the transaction information for the file.

-sqlrefresh

Preserves existing information about the table (synonyms, grants, etc.) when relinking into SQL.

Usage:

```
   -sqlrefresh file xfd_file database_name
               [-symb=table_name] [-prefix=table_prefix]
               [-owner=user_name] [-conv=convention_ID]
               [-rule=rules_file]
```

Description:

The **ctutil** *-sqlrefresh* command is very similar to the *-sqlize* command. The *-sqlrefresh* command has these properties:

1. It can be used only if the table was already sqlized.
2. It refreshes the table structure maintaining the existing grants and synonyms.
3. It does not support the "public" option available in *-sqlize* since authorizations are kept from the original sqlize and grant occurred in SQL.

Note: This is an administrative operation and therefore must be performed with caution. Do not attempt to use this operation on a file that is open. The <instance user> associated with the file for a sqlrefresh operation must have either DBA or RESOURCE SQL privileges in order for the operation to succeed or a c-treeRTG error 456 (group access denied) is returned. While strongly discouraged, backward compatibility is provided with this configuration option: `COMPATIBILITY SQLIMPORT_ADMIN_PASSWORD (https://docs.faircom.com/doc/ctserver/compatibility-sqlimport-admin-password-config.htm)`.

-test

Check the configuration and connection to servers. The syntaxes are:

```
ctutil -test config [file]
ctutil -test connect
ctutil -test filerules
```

- Running **ctutil** *-test config* option checks the configuration. If file is specified, it also checks which configuration instance matches the specified file.
- Running **ctutil** *-test connect* checks that all servers defined in the configuration (with the `<instance server>` attribute) are reachable.
- Running **ctutil** *-test filerules* will print the file rules in the order they will be considered by FairCom RTG at runtime when matching a filename:

```
ctutil -test filerules
Initialized from (ctree.conf)

<file name="*" dir="mydir" casesensitive="yes" type="*"/>
<file name="myfil" dir="*" casesensitive="yes" type="*"/>
<file name="*" dir="*" casesensitive="yes" priority="-32767" type="*"/>

Operation completed successfully.
```

Note: Running the -test command with no additional specifications will run the -test config command by default.

Obsolete Commands Removed

This modification removes support for some obsolete **ctutil** commands. The list of deprecated commands is:

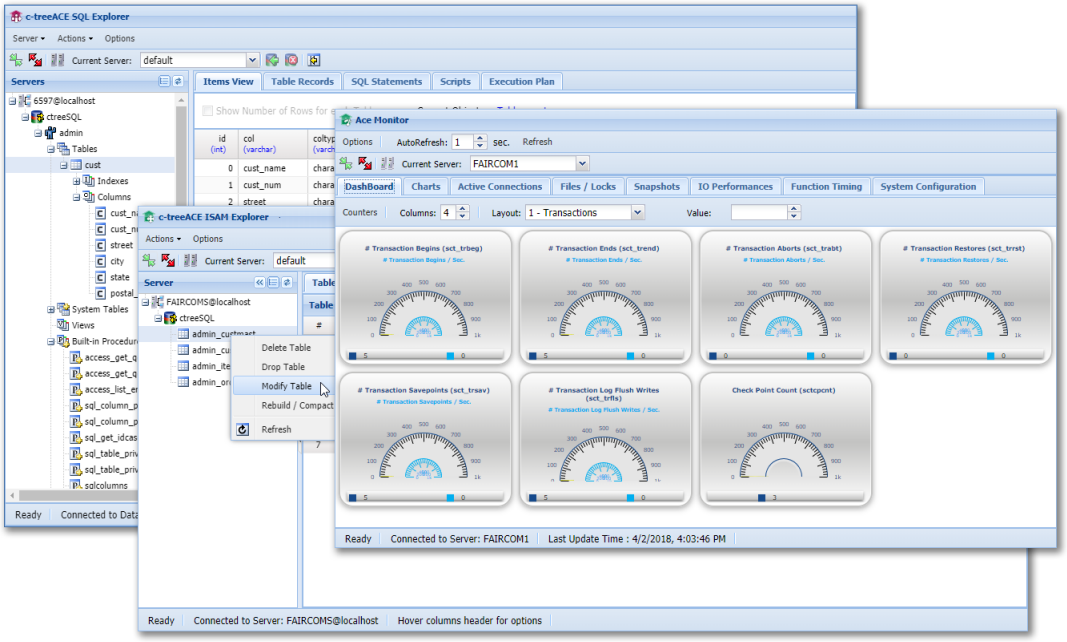
- *-param* (display parameter file) is obsolete.
- *-loadtext* (import from line sequential file) has been replaced by the *-t* option of the *-load* command.
- *-unloadtext* (export from line sequential file) has been replaced by the *-t* option of the *-unload* command.

11.2 Web-Based GUI Tools

FairCom is modernizing our tool suite in a web browser format. The following tools are now immediately at your fingertips from any browser connection:

- SQL Explorer
- ISAM Explorer
- ACE Monitor

- Replication Manager



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 Browser-Based Administration Tools (https://docs.faircom.com/doc/faircom-graphical-tools/).

11.3 ctclosefile - Close Open Memory and ctKEEPOPEN files

It is easy to create memory files or set persisted files in a *ctKEEPOPEN* state. However, those files need to be closed at times and we do not have a utility to do so, only the **CloseCtFileByName()** API. This revision adds a simple utility, **ctclosefile**, allowing this functionality in a scripted environment.

ctclosefile will close a specified FairCom DB memory file or any file held open with a ctKEEPOPEN directive.

```
ctclosefile [-u user] [-p password] [-s server][-f filename] [-w file password ] {-a authfile}
```

Parameters:

-u - User ID

-p - User Password

-s - Server (default: FAIRCOMS@localhost)

-f - File name

-w - File password

-a - Authentication file name

11.4 startserver and stopserver Scripts

Scripts have been added to the FairCom RTG product to start and stop the server. These scripts, **startserver** and **stopserver**, are equivalent to the FairCom DB scripts Unix sh (shell) scripts named **startace** and **stopace**.

11.5 ctinfo is Now Included in FairCom RTG

The **ctinfo** utility is now included in the FairCom RTG package. This utility retrieves *IFIL* and *DODA* structures from a FairCom DB file (and other header information).

11.6 ctfixdupscan - Detect and Fix Files that Suffer from File Definition Errors

New functionality has been added to the **ctutil** command line utility to detect and fix files that suffer from file definition errors that can cause the **dupscan** workaround to not function properly. To trigger this new functionality, rename a copy of the **ctutil** executable to **ctfixdupscan**. Executing this renamed copy of **ctutil** will cause it to function as an *ad-hoc* utility to detect and fix file definition errors. The syntax of the command-line tool is as follows:

```
ctfixdupscan file [-fix]
```

where:

- *file* - File name without extension
- *-fix* - Fix error by copying the file in place

By default, the program checks if the specified *file* suffers from file definition errors. If the optional *-fix* parameter is specified after the file name, the file is fixed by copying it in place record-by-record.

11.7 ctcmpcif - IFIL-based Compact Utility Included

The **ctcmpcif** utility has been added to the command-line utilities. This utility can be used to encrypt and decrypt c-tree data files.

Operational Model:

- Client
- Standalone
- FairCom DB Standalone SQL Service

Standalone Usage

```
ctcmpcif <data file name> <user> <pass> <server> <-sectors> <options>
```

Where:

- *<-sectors>* - The sector size to use. Enter a dash followed by the size of the sectors. The sector parameter is especially useful if you need to adjust a file's *PAGE_SIZE* to match the FairCom Server. This option must go after *<server>*. The sector size is 256 is in the example below:
  `ctcmpcif  xx8.dat ADMIN ADMIN FAIRCOMSMS -256`

`<options>` can be:

- *-callback* - Enable notifications of compact progress.
- *-compress* - Create the compacted data file with compression (defaults to zlib library if not specified in server configuration).
- *-encrypt=cipher* - Create the compacted file using the specified cipher. See the encryption sections below.
  Accepted values:
  - AES: aes16, aes24, or aes32
  - Blowfish: blf8, blf9, ..., or blf56
  - DES: des8, des16, or des24
  - none
- *-flexrec* - (Supported in V11.5 and later) Add Hot Alter Table support to the file.
- *-idxseg=<M>@<S>* - If creating an index, use *M* automatic segments of size *S*. *S*ize is specified in megabytes, or you can specify *MB* or *GB* as a suffix. (Example: *-idxseg=10@1GB*)
- *-local* - causes the utility to use a standalone (local side) connection instead of a client connection when linked with a LOCLIB library.
- *-nocompress* - Create the compacted data file without compression.
- *-noflexrec* - (Supported in V11.5 and later) Remove Hot Alter Table support from the file.
- *-oldsec* - Use the utility's original security attribute assignment.
- *-online* - (In V13 and later) Defragment the file in ctSHARED mode. Online compact is not allowed for changing encryption or compression settings.
- *-purge* - Delete records that have illegal duplicate key values.
- *-redosrl* - Reassign serial numbers to the records in the compacted file instead of copying the values from the original file.
- *-repairdata* - Attempt repair if damaged data records are detected.
- *-sortmem=<N>* - Set the size of sort memory to *N* KB in non-client builds. This option can be used to increase the rebuild speed for large files.
- *-temppath=<temporary_path>* - Use the specified directory for temporary files.
- *-updifil* - Update IFIL resources in the data file.
- *-x8* - Use extended create blocks read from data and index files.

Client-Side Usage

```
# ctcmpcif DataFileName [-purge] [-updifil] [<UserId>]
```

```
[<UserPassword> [<ServerName>]]
```

Where:

- *-purge -* purge duplicate records.
- *-updifil* - update *IFIL* resources in the data file.

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

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

Prior to FairCom DB V9.3, several option values were defined such as *updateIFIL*, *purgeIFIL*, and *badpartIFIL*, that can be specified. These values are specified by adding them together. For example: *myifil.tfilno* = *updateIFIL + purgeIFIL*. A new approach simplifies checking of these options. The following option values are now specified in *ctport.h*:

```
#define updateIFILoption     0x0002
#define purgeIFILoption      0x0004
#define badpartIFILoption    0x0008
#define redosrlIFILoption    0x0010
```

These values can now be OR-ed together and the negative of the result stored in the *tfilno* field of the *IFIL* structure passed to the compact or rebuild function. For example, to indicate that you want to assign new serial numbers, do the following:

```
myifil.tfilno = -redosrlIFILoption;
CMPIFIL(&myifil);
```

If you want to also use more than one option when compacting or rebuilding a file, OR them in to the value and negate the result. For example,

```
/* assign new serial numbers and update IFIL resource. */
myifil.tfilno = -(redosrlIFILoption | updateIFILoption);
CMPIFIL(&myifil);
```

ctcmpcif will attempt to open files of any page size definition. In addition, in standalone mode, it will create the new file based on the provided page size.

Note: This tool does not compact partitioned files. If you attempt to compact a partitioned file, the tool will return **PCMP_ERR** (954, compacting partitioned file not supported).

Changing Encryption Attributes

In V11 and later, the compact utility can optionally change the encryption attributes. To use this option, OR in the *setencryptIFILoption* bit into the *tfilno* field of the IFIL structure whose address you pass to the compact API function. When using this and other options, remember to negate the *tfilno* value after you OR in the options. For example:

```
myifil.tfilno = -(redosrlIFILoption | setencryptIFILoption);
CMPIFIL(&myifil);
```

Specifying the Encryption Cipher

In V11 and later, the **ctcmpcif** utility supports an option to specify the encryption cipher for the data and index files created by the compact operation. Usage:

-encrypt=cipher - Create the compacted file using the specified cipher:

- for AES, use aes16, aes24, or aes32
- for Blowfish, use blf8, blf9, ..., or blf56
- for DES, use des8, des16, or des24
- for Twofish, use twf16, twf24, or twf32
- for no encryption, use none

Note: If an index file does not exist, the original data file's encryption attributes are used when creating that index file.

To change the encryption attributes of a file using this ctcmpcif compact utility or the compact API function from a client, you must add the option `CHANGE_ENCRYPTION_ON_COMPACT YES` to *ctsrvr.cfg*. Otherwise, the operation fails with error **NSUP_ERR** (454, not supported).

Environment Variable to Enable Advanced Encryption

In V11.5 and later, c-tree supports enabling advanced encryption at run time using an environment variable. Set the environment variable `CTREE_ADVANCED_ENCRYPTION` to `YES` to enable advanced encryption if it is supported. This environment variable can be used to allow c-tree utilities to enable advanced encryption even if they haven't been updated yet to automatically enable advanced encryption when needed. Examples include the rebuild and compact utilities, **ctrbldif** and **ctcmpcif**.

Note: If c-tree does not support advanced encryption and this environment variable is set, the c-tree initialization will fail.

Note: In order to use advanced encryption, a master key must be supplied. The utility will prompt for the master key, or a master key file can be created like this:

1: create master key file:

```
ctcpyf -k mykey -s mykey.fkf
```

2: set environment variable to the name of the master key file:

```
CTREE_MASTER_KEY_FILE=mykey.fkf
```

If using these steps, then the utility will read the master key from the file instead of prompting for the key.

Considerations

- Even without the *-x8* option, some extended create block settings such as the huge file, extended header, and 6-byte transaction number options, are always applied to new files.
- The *-x8* option requires all associated index files that are referenced in the data file's *IFIL* structure to exist, as the utilities use **OpenIFile()** to open the data file and all associated index files to read the extended create block values.

When an application calls the standalone version of the *Xtd8* file compact or rebuild functions (for example, **CMPIFILX8()** or **RBLIFILX8()**), index files created by the compact or rebuild no longer

have the 6-byte transaction number attribute enabled if the specified extended create block's *x8mode* field has the *ctNO6BTRAN* bit set.

- A rebuild or compact will fail with the new error code **TFLN_ERR** (943) when a client library that supports the new rebuild or compact option format attempts to use this feature with a FairCom Server that does not support this new format.
- In V11 and later, the compact and rebuild API functions preserve the original files' encryption attributes by default. If the index files don't exist when compact/rebuild are called, they will be created with the data file's encryption attributes.
- In V11 and later, the compact utility always deletes and recreates the index files after saving the resources from the original index file. This provides the expected behavior of (possibly) reducing the size of the index file.

See also

- *File Recovery (https://docs.faircom.com/doc/ctreeplus/30588.htm) in the FairCom DB Developer's Guide*
- *Updates in handling of security attributes*
- *Preventing Possible Data Loss with Compact & Rebuild Operations*

11.8 ctfdump Utility Extended with !RECOVER_DETAILS Option for Progress Notifications

In FairCom DB V12 and FairCom RTG V3, a new `!RECOVER_DETAILS` command-line argument, instructs the forward roll utility, **ctfdmp**, to provide progress notifications. This argument enables additional logging to *CTSTATUS.FCS*. The main progress indicator for a forward roll are messages such as:

```
First stage: Transaction Undo: A reverse log scan from the most recent log (Log 7 in this case) to Start
point (Log 2 in this case)
- User# 00001 tranund: reverse scanning log 7, undos=313
- User# 00001 tranund: reverse scanning log 6, undos=219
- User# 00001 tranund: reverse scanning log 5, undos=106
- User# 00001 tranund: reverse scanning log 4, undos=60
- User# 00001 tranund: reverse scanning log 3, undos=34
- User# 00001 tranund: reverse scanning log 2, undos=7

Second stage: Transaction Redo: A forward log scan applies changes from transactions beginning at the Start
point until the end point is reached.
- User# 00001 tranred: scanning log 2, redos=33096
- User# 00001 tranred: scanning log 3, redos=32646
- User# 00001 tranred: scanning log 4, redos=28297
- User# 00001 tranred: scanning log 5, redos=26009
- User# 00001 tranred: scanning log 6, redos=20087
- User# 00001 tranred: scanning log 7, redos=8336
```

11.9 ctldmp Utility Enhanced to Display File ID Values

The **ctldmp** utility now displays the file ID component header values that are included in file open, close, and file ID reassignment log entries. Example output:

```
LOGPOS:01-00015ae2x #0:000015 P005ea054b5-00x  F0022-600 T27 U11 A0000x L116 042 OPNTRAN
00000000000000000000000000000000005313b5a5445455454452445000000000000000000000
10000000000000000000000000000000a0006429440e462b3414544e6330000000000000000000
.................................V4.9.T.^DFRKSTATEDT.FCS...................
        fileid=10 (0x0000000a)
        servid=957494358 (0x39123456)
        timeid=1587565748 (0x5ea054b4)
```

11.10 ctfileid - Assign a New Unique ID to a Data or Index File

The **ctutil** *-fileid* command resets the unique file ID of a file that has been copied at the system level. The syntax is:

```
ctutil -fileid file
```

This support accompanies `LOG_FILE_ID_CHANGE` for reassigning file IDs discussed here: *Improved performance reassigning transaction-controlled file's ID (page 30)* . This command applies the FairCom DB **PUTHDR()** (https://docs.faircom.com/doc**/ctreeplus/updateheader.htm**) mode *ctUNIQIDhdr* to
assign a unique file ID to the specified file.

For c-treeRTG, if run against a server that does not support the *ctUNIQIDhdr* mode, **ctutil** fails with error message "**Client/server incompatibility**" and the following error message is logged: "ERROR 37:-4:0 server does not support fileid reset".

See Also:

- *Copying Server-Controlled Files*
- *ctfileid - Update File IDs* (https://docs.faircom.com/doc/*ctreeplus/ctfileid-util.htm*)

11.11 ctstat - Display Log Save Time Delta Values

When the *-d* option is used, the **ctstat** utility now calculates and outputs the average log save time (the average time for transaction log writes) in microseconds for each **ctstat** monitoring interval. Below is an example showing sample output. See the rightmost value, "log save":

```
# ctstat -vas -d -u ADMIN -p ADMIN -s FAIRCOMS

 cache           disk i/o  files  conn  locks         transactions         log
d%h i%h d%u i%u  r/s  w/s   cur    cur  cur l%h dead  act  t/s  r/t  w/t  save
 99 100   1   0    0    0    24      1    1  99    0    0    0    0    0     0
100 100   1   0    1    1    24      1    1  99    0    0    0    0    0   195
100 100   1   0    0    0    24      1    1  99    0    0    0    0    0     0
 99 100   1   0  620  740    29      9    6  99    0    4  739    1    1    85
 99 100   2   1 4545 5844    29      9    7  99    0    5 6088    1    1    79
 99 100   2   2 4244 5881    29      9    8  99    0    7 6092    1    1    81
 99 100   3   2 4074 9495    29      9   13  99    0    6 5674    1    2    79
 99 100   4   3 4024 9898    29      9    5  99    0    5 5505    1    2    80
 99 100   4   4 3773 9787    29      9   10  99    0    6 5507    1    2    80
 99 100   5   4 3927 9838    29      9   10  99    0    6 5469    1    2    80
```

11.12 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

11.13 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.

11.14 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

11.15 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

11.16 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.

11.17 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

11.18 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).

11.19 Diagnostic Session Recording

cttrap (https://docs.faircom.com/doc**/ctreeplus/cttrap-util.htm**) utility supports displaying TRAPCOMM log contents.

12. Plug-ins and Callbacks

The FairCom RTG Server can be expanded with two new capabilities available in V3:

- **Plug-in technology** makes it easy for integration with your COBOL data through a variety of new interfaces detailed below;
- **Callbacks** are now documented in their own book that makes it possible to see how you can expand callback hooks to gain access to your COBOL data.

12.1 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.

12.2 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/**)

13. Upgrade Steps

Your existing FairCom RTG system can be upgraded to the latest version with very few changes. It is important to follow the procedures in this section to upgrade in a safe manner.

Compatibility Note: FairCom RTG features a new, streamlined folder structure. This change results in a new location for the server and other FairCom RTG components such as the utilities. Any existing scripts that rely on these locations will need to be updated. For more, see *New FairCom RTG footprint* (page 101).

Upgrade to the Latest FairCom RTG V3 Database Engine

While FairCom attempts to maintain backward compatibility when at all possible, transaction logs from earlier versions are not always compatible with newer FairCom RTG formats. For example, FairCom RTG V3 is built upon the latest FairCom Server, which introduces changes in the transaction log to accommodate new capabilities. Since backward compatibility is affected with this upgrade, then all of the steps in the following upgrade procedure are required.

Note: Existing data and index files are not affected by this update.

It is easy to install and use FairCom RTG with your existing files by removing prior transaction logs in a safe manner. Follow these easy steps, which are appropriate any time you are upgrading a c-tree installation:

1. Have all clients cleanly exit from the existing the c-tree Server.
2. Perform a controlled shutdown of the c-tree Server with your normal shutdown sequence. Verify the c-tree Server shutdown was successful by looking for the following "Server shutdown completed" in *CTSTATUS.FCS*:
   ```
   - User# 00023 Server shutdown completed
   ```
3. Block the ability of any clients to attach to the c-tree Server.
4. Restart your existing c-tree 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 your existing c-tree Server.
6. Remove this list of existing transaction logs and associated files (*L*.FCS*, *S*.FCS*, *D*.FCS*, and *I*.FCS*).
7. Copy your new FairCom Server executable (**faircom.exe**) and any companion *.dll* (*.so* on Linux / Unix) into the existing *server* directory. Be careful not to overwrite any of your configuration files (**.cfg* and **.conf*) and your *FAIRCOM.FCS* file.

> **Note**: It is always advisable to use the most recent matching client version and utilities with your FairCom RTG server version. By updating the client as well as the server you will be able to leverage the new capabilities that otherwise would not be available

8. Unblock the ability of any clients to attach.
9. Start FairCom DB in your usual manner and begin accessing your existing data.
10. **Optional - for SQL users**: Sqlize your tables using the **ctutil** *-sqlize* utility or **ctutil** *-sqllink*. The new version of FairCom DB SQL Explorer provides new features, such as highlighting sqlized tables in different colors, checking for bad records, and adding SQL indexes (see *FairCom DB SQL Explorer*). If you want to use these features, you will need to sqlize after upgrading to FairCom RTG V3 **even if you have sqlized in an earlier version of FairCom RTG**.

FairCom RTG Component Upgrade

You need to use the new FairCom RTG driver included with this release. For ACCUCOBOL users, this also means compiling a new *ctreeacu.c* module. Refer to this location if you need a refresher for how to do this. See the setup section for your compiler in the *FairCom RTG User's Guide* (https://docs.faircom.com/doc/*ctcobol/*) for details.

SQL Driver Upgrades

FairCom RTG includes a collection of SQL drivers to allow relational access through a variety of interfaces.

- JDBC drivers
- ODBC drivers
- PHP modules
- ADO.NET drivers
- Stored Procedure frameworks

To upgrade the drivers, simply uninstall the existing driver, and then run the new Driver Installation program provided by FairCom. If no standalone driver is provided for your driver, then copy in the new libraries provided in this FairCom RTG package.

13.1 New FairCom RTG Footprint & Upgrade

The FairCom RTG product footprints for COBOL and BTRV have been updated. The directory structure has been updated to make the product easier to use. The primary change is the **ctreesql.exe** (**ctreesql** on Unix) database engine has been moved up one directory, and has been renamed to **faircom.exe** (**faircom** on Unix/Linux). Previous versions had a directory structure *server\sql*. Going forward, the *sql* directory has been removed.

> **Compatibility Note**: This change results in a new location for the server. Any scripts that rely on these locations or the executable name will need to be updated.

This build of FairCom RTG includes the new replication technology known as Replication Manager. See *FairCom RTG Replication* for information about the product.

Full documentation for this product is available here: *Replication Manager User Guide* (https://docs.faircom.com/doc/*replication-manager/*)

13.2 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/
MQTT Explorer - SSL	8443	cthttpd.json	V12.5: "https_port": 8443 V12: "listening_https_port": 8443	https://localhost:8443/mq/ 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/sqlexplorer/
ACE Monitor - SSL	8443	cthttpd.json	V12.5: "https_port": 8443 V12: "listening_https_port": 8443	https://localhost:8443/monitor/ 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

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
FairCom DB API, 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*

13.3 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).**

13.4 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.*

14. Compatibility Notes

With any new release, certain features preclude prior default functionality. FairCom RTG V3 introduces several new advances that may impact your prior c-tree experience. Notably in this release is a new default folder layout (page 109) simplifying access to c-tree components. A major change to default IFIL path handling is also included. Be sure to review this entire list and verify any specific changes that apply to your usage.

As always, V3 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 as described in **FairCom RTG V3 Upgrade & Compatibility** (page 100). Contact your local FairCom support team with any questions about compatibility and suggested upgrade steps.

14.1 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`.

14.2 Max Key Segments Increased

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.

14.3 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.

14.4 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.

14.5 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.

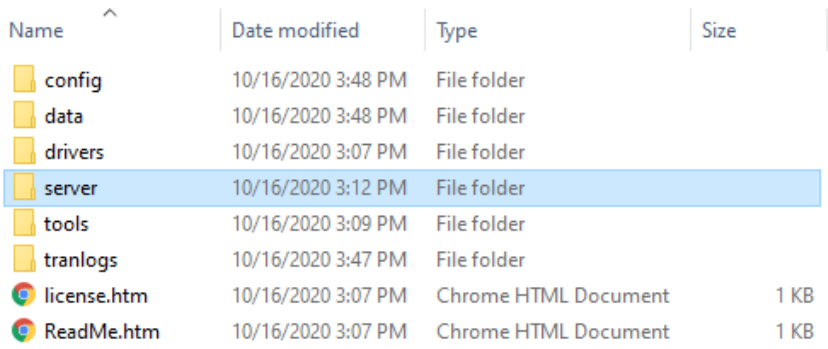
MAX_DFRIDX_LOGS

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.

14.6 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 111) 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, node.js (SQL and direct record oriented) and Python drivers here. This is equivalent to the */sdk* folder in prior releases.

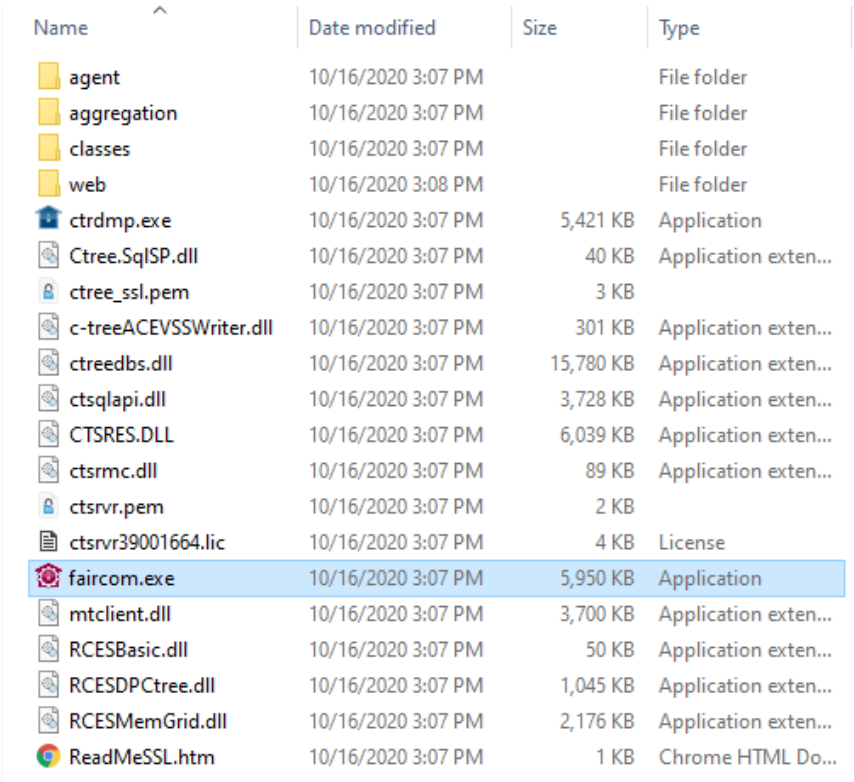
Inside this area we also have consolidated prior pre-compiled libraries, necessary project includes and binaries. These were previously found in */lib* and */bin* folders.

The embeddable server model ("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 98) features. Data aggregation, replication agent, and the web apps 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!

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)

14.7 <config> Attribute <config prev12filematch> for File Matching Rules

In FairCom RTG V3 and later, new simplified file matching rules are used to determine which rule takes precedence when more than one rule applies to a file.

An attribute has been added to the `<config>` element to revert to the old rules for compatibility:

1. The `<config>` attribute `<config prev12filematch>` designates that FairCom RTG should use the old rules for file matching. This attribute is a boolean that defaults to `'no'`.
2. If `<config prev12filematch="yes">` is specified, the original file matching algorithm is used in place of the newer, simplified rules.

Details on the new (default for V3) and old files rules (default for V2 and prior) are here: *File Matching Precedence* (https://docs.faircom.com/doc/*ctcobol/_file_.htm#o71510*)

14.8 Improved IFIL Path Handling

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.

14.9 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)
```

14.10 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.

14.11 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 */
```

14.12 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.

14.13 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.

14.14 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
}
```

14.15 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).

14.16 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.

14.17 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.

14.18 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.

15. FairCom RTG - Migrating and SQLizing Data

15.1 Millisecond Time Support Added

The FairCom RTG time and datetime data types are now mapped to FairCom DB API types with millisecond support. Prior to this modification, the FairCom RTG SQL callback was mapping time and datetime to FairCom DB API types with no millisecond support. With this change, the milliseconds support means that the "TT" part of the format (which was ignored before) will be considered.

This should not affect backward compatibility for users not using the "T" in the time format, and it will fix the problem of ignoring hundredths of seconds for those using 'T'.

15.2 Automatic sqlize Logic Allows an XFD and/or XDD to Be Specified

The automatic **sqlize** logic has been changed so that the *xfd* and *xdd* options are no longer synonymous. If a `<sqlize xdd="xxx">` attribute is specified, it looks for a file with *.xdd* file extension. If a `<sqlize xfd="xxx">` attribute is specified, it looks for a file with *.xfd* file extension. If both attributes are specified, the logic first looks for an XDD file and if the file is not found, sqlize falls back to looking for an XFD file.

Note: This modification is a behavior change as `<sqlize xdd>` is no longer a synonym of `<sqlize xfd>`.

15.3 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
```

15.4 COBOL Date Baseline Can Be Set to Julian Starting Date of Dec 31, 1600

COBOL has no "date" type, however it has a standard function called **integer_of_date**, which takes a COBOL number of the form `YYYYMMDD` and generates an integer that is the number of days since Dec. 31 1600 (therefore a Julian date). When linking such a date into SQL, the julianBase would be "16001231". However, such a julianBase cannot be specified because the value needs to be in c-tree valid date range, which is not earlier than March 3 1700.

It is now possible to specify in the XDD `julianBase='integer-of-date'` (`'integer-of-date'` must be lowercase!) to indicate values stored in the record result from an integer-of-date function.

15.5 xddgen - New Configuration Option max-fixed-record-len

When RM/COBOL is configured to use Btrieve as the file handler, the creation of a fixed-length file with record size over 4082 actually creates a variable-length file with a minimum record length of 4082 and maximum record length of whatever is the record size. This can cause a problem when an XDD is generated and the file is SQLized or the XDD is used to re-create the file.

Because RM/COBOL does not have a way of generating a "schema" that can be translated into an XDD, **xddgen** is the only way to produce an XDD.

Given the FD and the SL with a record size greater than 4082, **xddgen** produces a XDD containing minRecLen equal to maxRecLen. When this XDD is used to sqlize the file created by RM/COBOL, the SQL callback fails because the definition is not correct (the file is variable with a fixed-length of 4082). When used to generate a file (**ctutil** *-make*), the RM/COBOL runtime fails with error **39.09** because the definition does not match.

xddgen has no way to know about this limit on the minRecLen unless instructed so. Therefore, a new configuration parameter has been added:

```
max-fixed-record-len
```

The default is 0, meaning no limit. When set to a value other than 0, it is used to limit the minRecLen value when generating the XDD.

To use this keyword, create your own **xddgen** configuration file with content similar to:

```
include "default.conf"
max-fixed-record-len: 4082
```

15.6 xddgen - New Record Size Checks and Warnings

When the FD does not explicitly define a record size (using COBOL-specific syntax), **xddgen** determines the minimum and maximum record length by looking at the minimum and maximum record size of the various records layouts (level 01).

When COBOL does not specify a record length, the record is always fixed-length, but the records layout may have different sizes and so the generated XDD have different minimum and maximum lengths, which then (depending on how these are set) fail the check against the physical file record length causing a 4110 error (**CALLBACK_2**).

The logic has been changed so that, if the record length is not specified, it considers the record as fixed-length and uses the first record definition length as the "base length." It then checks each record size against the specified or calculated record minimum and maximum lengths and prints a warning if its size is outside of the expected range.

The XDD will be generated using the minimum and the maximum record lengths found in the various record definitions (schemas) as it always was.

If the XDD is going to generate information with different minimum and maximum record lengths (thus implying a vlen file) but the COBOL source did not specify the record as variable length, a warning message is printed. **It is recommended to modify the COBOL source to resolve these warnings.**

It was also identified that the size of COMP-4 for default AcuCobol configuration was not correct. The "binary-size" has been updated to the proper setting of "2-4-8" in *acu.conf* and *acu-Ds.conf*.

15.7 Standalone Support for ctmigra

FairCom RTG supports the FairCom Standalone operational model for specific use cases. This differs from the default client/server model by not using the FairCom RTG Database Server (**faircom.exe**, or **ctreesql.exe** in versions prior to V3). By not going through the FairCom RTG Database Server, the database I/O can at times be faster, typically when there is only a single instance performing database I/O. If you are using multiple versions of **ctmigra** to migrate several files at the same time, then we don’t recommend using the Standalone model.

ctmigra --local=LIBRARY Option

The new **ctmigra** option *--local=LIBRARY* allows setting the FairCom RTG Standalone library where *library* is a FairCom RTG Standalone library.

The *--local* switch instructs **ctmigra** to work in the Standalone operational model. When this switch is set, the following switches are all disabled:

-n (or *-N*) used for specifying the FairCom RTG Server name

-u (or *-U*) for specifying the user ID

-p (or *-P*) used for specifying the user ID password

ctmigra --local-sect Option

This modification introduces a new option in **ctmigra** to set the number of node sectors when using the option *--local*. The new option is *--local-sect=SECT* where *SECT* is the number of node sectors to use in the *<localinstance sect>* attribute of the **ctmigra** auto-generated FairCom RTG configuration file.

The option *--local-sect* is indicated when the file migrated by the local engine (**ctmigra** option *--local*) will be open by a c-tree Server configured with a `PAGE_SIZE` set to a higher value then the default 8192 (8KB).

If *--local-sect* is not specified, the default value of `<localinstance sect>` is used, which is 64 resulting in a node of 8192 bytes (64 * 128 bytes) matching the default `PAGE_SIZE` of the c-tree Server.

If, for example, the c-tree Server has a `PAGE_SIZE` of *32768*, the **ctmigra** *--local* command should also include *--local-sect=256* so the local engine will create files that can be open by c-tree Server.

When this option is set, the following are all disabled:

- *-n* (or *-N*) used for specifying the FairCom RTG Server name
- *-u* (or *-U*) for specifying the user ID
- *-p* (or *-P*) used for specifying the user ID password

The option *--local-sect=SECT* where *SECT* is the number of node sectors can be used to specify the node size used to create the c-tree file.

The number of node sectors to use when performing data migration is important to ensure the data file created during the **ctmigra** process are of the same `PAGE_SIZE` being used by the FairCom RTG Server. The `SECT` setting provides a dynamic method for setting the matching FairCom RTG Server `PAGE_SIZE` setting. Note this equation: `SECT*128 bytes = PAGE_SIZE`. The default `PAGE_SIZE` for FairCom RTG V3 is now 32,768, therefore the default `SECT` value is 256. Here are `SECT` values for setting the more common `PAGE_SIZE` settings:

SECT	PAGE_SIZE	
64	8192	Default FairCom RTG < V3
128	16384	
256	32768	Default FairCom RTG V3
512	65536	

If *--local-sect* is not specified, the default value of `<localinstance sect>` is used, which is 256 for V3, and 64 for FairCom RTG versions prior to V3.

If, for example, the c-tree Server has a `PAGE_SIZE` of 16384, the **ctmigra** *--local* command should also include *--local-sect=128* so the local engine will create files that can be opened by c-tree Server.

<localinstance> bufs, dbufs, sect Value Check

The attributes `bufs` and `dbufs` allow setting c-tree standalone index and data number of cache buffers respectively. The maximum acceptable value is 32,767.

The `sect` attributes set the table node size (which should match the server `PAGE_SIZE` (https://docs.faircom.com/doc/`ctserver/page-size-config.htm`)). The maximum is 512.

15.8 FairCom RTG - Conversion Sample Updated

This modification introduces the following enhancements to the *mfconvert.cbl* sample program:

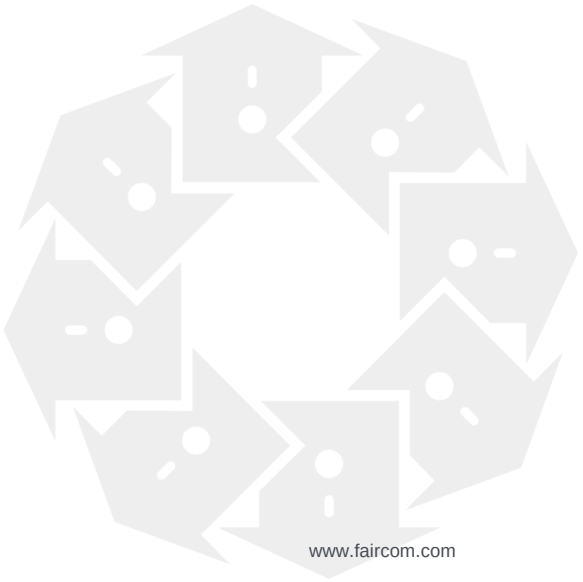
- **Return value** - The *mfconvert* program now returns a value of 0 if the conversion was successful, 1 if it failed for a generic issue, or the COBOL status code if a COBOL error occurred.
- **Usage display** - The *mfconvert* program now displays a usage message if it is run with fewer than 2 command-line parameters.

16. New Platforms

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

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*



17. FairCom RTG - BTRV Edition

The following changes apply to the BTRV Edition only. Be sure to note the "BTRV Specific" call outs in other locations throughout this book. Searching for BTRV will help you easily identify those locations.

17.1 FairCom RTG BTRV Login/Logout Operation

The BTRV Login/Logout operation (BTRV opcode 78) is now supported.

The BTRV application must call the BTRV Login API passing a URI in the BTRV standard format:

```
access_method://user@host/dbname?parameters
```

where:

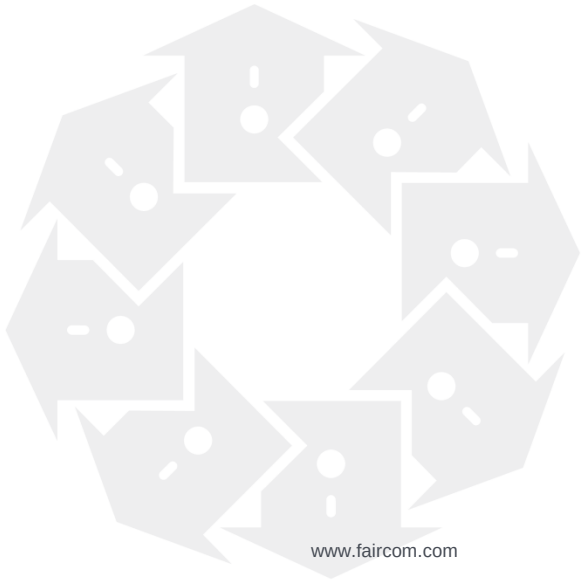
- *access_method* is "btrv"
- *user* is the c-tree Server user ID
- *dbname* is the c-tree Server name
- user password is specified in parameters as "pwd=password"

If the Login operation returns successfully, all subsequent operations will be operated on the specified server until a Logout operation is called.

17.2 BTRV Extended Index Types

Support has been added to FairCom RTG BTRV for extended key types on the following BTRV types:

- STRING
- INTEGER
- IEEE
- DATE
- TIME
- DECIMAL
- MONEY
- NUMERIC
- ZSTRING
- UNSIGNED_BINARY
- AUTOINCREMENT
- BIT
- NUMERICSTS



- NUMERICSA
- CURRENCY
- NUMERICSLB
- NUMERICSLS
- NUMERICSTB
- NULL_INDICATOR

BTRV transactional type column = c-tree ISAM engine.

Relational type = c-tree SQL engine.

BTRV Exclusive transaction = Means only one user. c-tree doesn't have exclusive transaction; similar to single user.

17.3 Support for Exclusive Transactions in FairCom RTG BTRV

FairCom introduces exclusive transaction support required for better compliance with the Btrieve API. A file accessed within an exclusive transaction is now locked for the entire duration of the transaction.

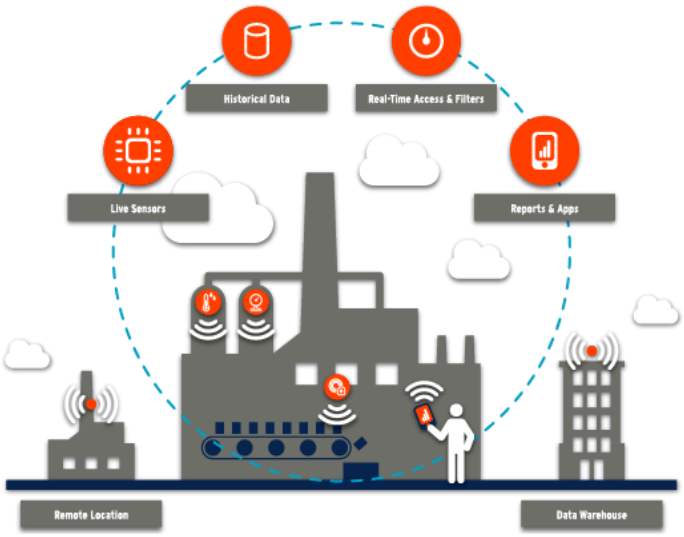
17.4 Support for BTRV Create Index Operation

This release introduces support for creating new indexes on existing files (op code 31) in FairCom RTG BTRV.

18. 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.

18.1 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

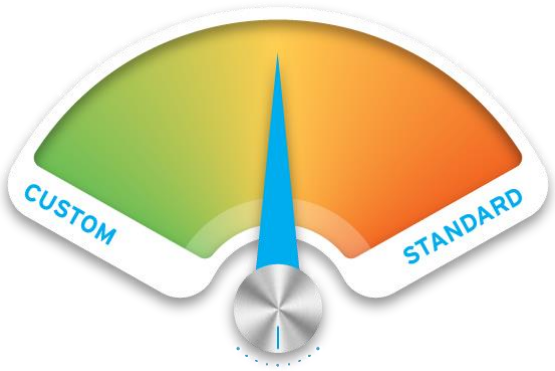
More information on FairCom Edge is available on our web site: FairCom Edge Product Page (http://www.faircom.com/products/c-treeedge-iot-database)

18.2 FairCom Database

The FairCom Database has a 40-year history of refinements that has made it the number one choice for developers who need to integrate with a high-performance database. This is the database engine that powers FairCom RTG. It is also available for use in any of your development projects, whether they be in C, C++, C#, VB, Python, Java, JavaScript, or Node-RED, FairCom DB can integrate with your application.

FairCom DB is ideal for large-scale, mission-critical, core-business applications that require performance, reliability, and scalability that cannot be achieved by other databases. It delivers predictable, high-velocity transactions and massively parallel big data analytics. It empowers developers with NoSQL APIs for processing binary data at machine speed and ANSI SQL for easy queries and analytics over the same binary data.

FairCom DB gives you a Continuum of Control to achieve unprecedented performance with the lowest total cost of ownership (TCO). With FairCom DB, you are not forced to conform your needs to meet the limitations of the database. You can conform FairCom DB to meet your business needs, giving you the database you need to meet your core-business needs...fast, reliably and efficiently.



You don't conform to FairCom DB ... FairCom DB conforms to you.

More information on FairCom DB is available on our web site: FairCom Database Product Page (https://www.faircom.com/products/faircom-db)

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.

1/30/2025

19. Index