test link

Update Guide

c-treeACE V10.0 Update Guide

Audience

Developers

Subject

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

FairCom®

Contents

1. Introduction

"Scale UP to c-treeACE Version 10"

c-treeACE V10 - Another Major Milestone from the Precision Database Engineers

The FairCom team is pleased to present our tenth major edition of c-treeACE database technology. This release benefits from the extensive feedback from customers whose implementations you encounter every day. Whether you are using your credit card, shipping a package, picking up a lottery chance, or buying tickets for your favorite concert, you are supported by the power of c-treeACE—we work closely with the vendors of these commercial applications around the world.

FairCom's investment in this sophisticated database technology began over three decades ago. Today, the same core team retains this commitment to efficient technology. This longevity, consistency, stability, and level of expertise is evident in V10.

We would like to thank our customers for their support. We owe you the credit for pushing us with your real-world demands. Our engineering team enjoys the technical challenges and the benefits derived from improving our products while servicing you. If you have not experienced direct contact with our technical team, we encourage you to do so.

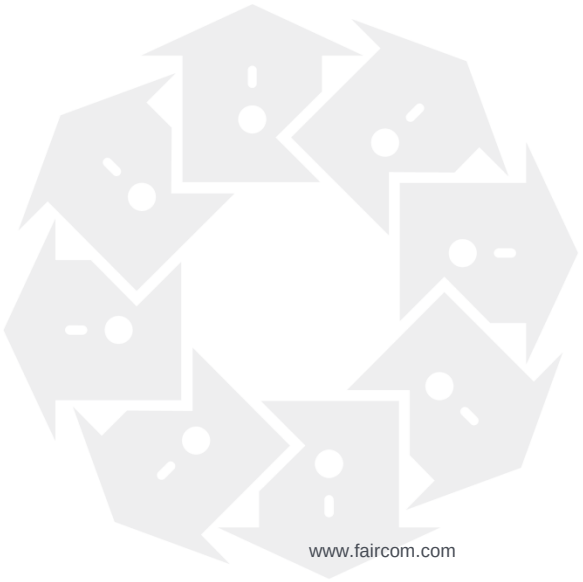
Precision engineering is what we do. FairCom's exacting engineering helps you deliver a precision product to your customers. FairCom provides a unique model not available from traditional database vendors. Many high-end production systems require this exceptional approach to get their job done. We start with proven high-quality database technology and the people who know this code. We add the ability for customers to engage in a close technical relationship with our experienced engineering team. The result is a precision engineered solution for your exacting needs. Your ability to work with us on special requirements results in insights to our products and better solutions for you. Whether the stipulation is more performance, an alternate data model, a port to a new platform, or your own unique specifications, we encourage you to take advantage of our engineering services.

This new release is packed with innovations, improvements, and enhancements.

So, whether you are a new c-treeACE user or a veteran code warrior, we encourage you to look at our latest efforts that "Scale Up" to the best work we have yet offered: c-treeACE V10.

We thank you for your business.

Sincerely,
The FairCom Team



2. V10.0 Highlights

This section presents an overview of the enhancements in V10. If you are upgrading from an earlier release, see the appendices for important compatibility information:

- Critical Production Updates (page 210)

- Notable Compatibility Changes (page 214)

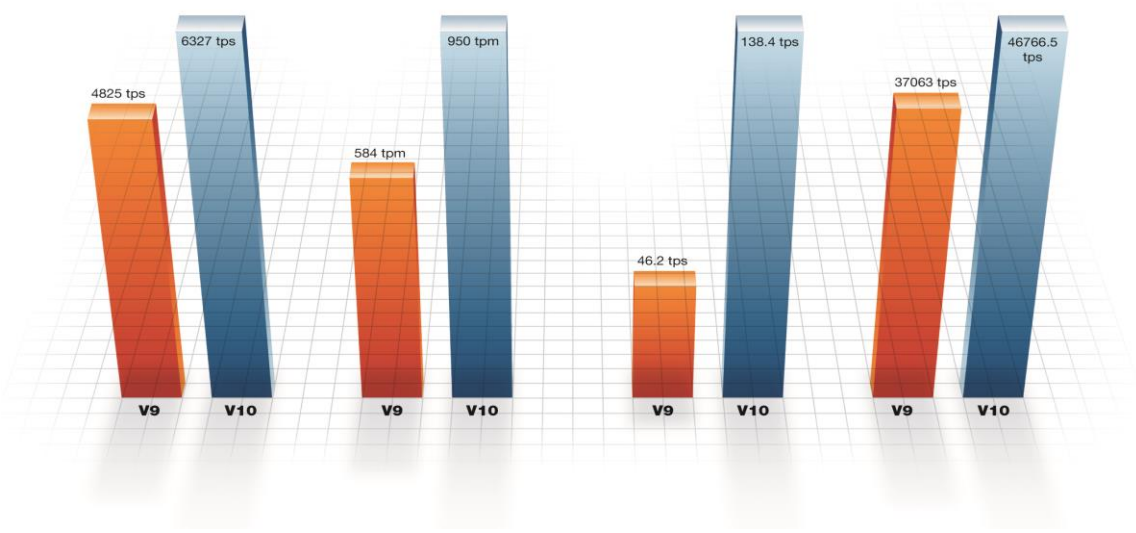
c-treeACE V10 features a newly improved, user-friendly activation mechanism. To activate it, see:

- IMPORTANT: New Licensing & Activation (page 23)

2.1 Performance

Scale UP to Enhanced Performance!

- Commit Read Lock optimizations
- Advanced hash lookups speed filename searches
- Unbuffered I/O bypasses file system cache
- Faster numbering of transactions
- Reduced memory suballocator contention scalability
- Reader/Writer Locks enhance synchronization efficiency
- Numerous SQL query optimizations
- Range search optimizations
- Unix shared-memory connections
- Reduced index node contention
- Critical Section Spin Count optimizations on Windows
- Reduced file header reads
- Intelligent file open/close logic for increased efficiency of file handlers
- Improved PreImage search of transaction hash entries



30% Faster Transaction Throughput	60% SQL Performance Improvement	200% Better Replication Throughput	26% Faster Read Performance Testing
Test Environment: FairCom's cttctx load test utility simulating a record add / read / delete sequence on 23 files with indices.	Test Environment: Benchmark Factory TPC-B	Test Environment: FairCom's cttctx load test simulating a record add/read/delete on 23 files with indices then applying those changes to a target server on the same hardware.	Test Environment: FairCom's cttctx load test utility simulating a record read sequence on 23 files with indices.
• 100 threads • 10,000 iterations • 1,000,000 transactions	• 20 threads • 3 minute run • 1.5GB File size	• 100 threads • 10,000 iterations • 1,000,000 transactions	• 100 threads • 10,000 iterations • 1,000,000 transactions

- Hardware: Dell PowerEdge, 16 Cores, 3.6 GHz, 32 GB RAM, Seagate ST3600057SS SCSI Hard Drive, 2 X 600GB, 15K rpm
- Windows Server 2008 R2 64-bit, Datacenter Service Pack 1

The considerable performance gains experienced with the V10 release are a result of astute profiling of the c-treeACE database engine internals. Scalability issues, bottlenecks, contention points, wait times, capacity sizes, as well as many other details were wholly critiqued through thorough profiling, testing, and extensive benchmarking. Additionally we encouraged and considered “real world” feedback from customers. We are excited with the results and encourage you to adopt this V10 technology as soon as possible and experience these rewards.

Later in these notes, the **Foundations of V10 Performance Gains** (page 130) section details the significant internal improvements within the c-treeACE core engine. Many are low-level internal changes and the combination of all these improvements resulted in overall higher c-treeACE performance with much improved scalability. The highlights are summarized below. The internal details are most applicable to those who wish to understand the depth behind the significant performance improvements in V10.

FairCom has subjected our latest releases to extensive testing in areas of scalability and performance to leverage technological advances in today’s operating systems and hardware platforms. We have achieved significant internal core engine performance improvements in the areas listed below.

Synchronization Objects - Semaphore/Mutex:

- Shared memory control
- Reduced synchronization redundancies
- Minimized internal contention of read/write queues
- Profiled memory file inserts
- Improved critical section spin count control
- Reduced node contention in index key lock management
- Efficiencies gained through native OS “reader/writer” synchronization support

Transaction Processing:

- Faster Automatic Recovery
- Faster Transaction Log processing
- Faster assignment of Transaction Numbers
- SAVE-POINT efficiency
- Transaction Preimage performance gains
- Dynamic Hash Bins: transaction entry search speed

File I/O:

- File Read-Only Open speed: Large number of open files
- File Open/Close overhead: Leave files open
- File name Hash List improvements
- Faster internal user file control block handling

Memory:

- Memory sub-allocator scalability improvements
- Memory sub-allocator adjustments for faster memory searches
- Memory File Configurable Hash Bins: Improved 64-bit addressing

Locking:

- Commit Read Locks: Significant performance gains; enhanced scalability
- Exclusive Files: No lock overhead
- Key-Level Lock optimizations

More Advances:

- User connection performance during a live backup: Dynamic Dump
- Estimate Key efficiencies
- Range search speed
- Cache efficiency: Unbuffered I/O to bypass OS's file system cache
- Read Header read: minimize need; increase speed

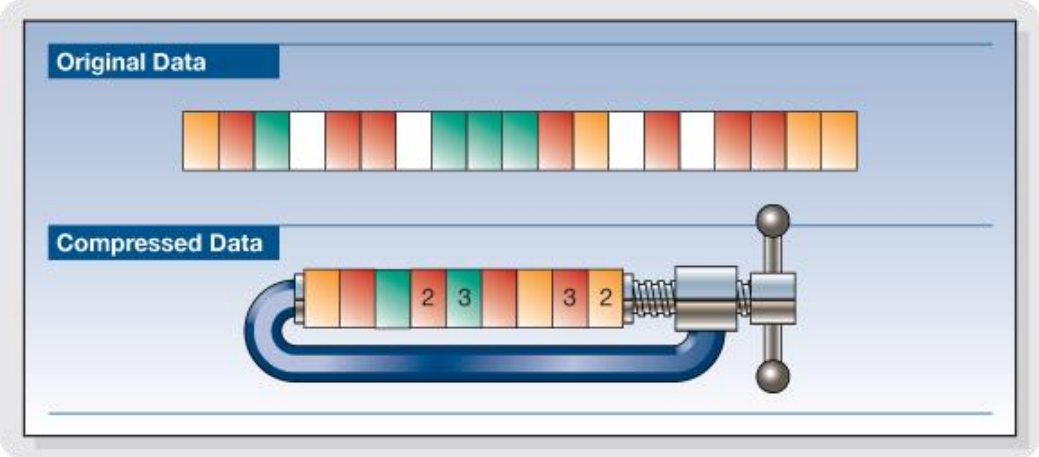
Need more proof? For a detailed list of the development work we have done to bring you enhanced performance, see the section titled **Foundations of V10 Performance Gains** (page 130).

2.2 File Management

Data Compression

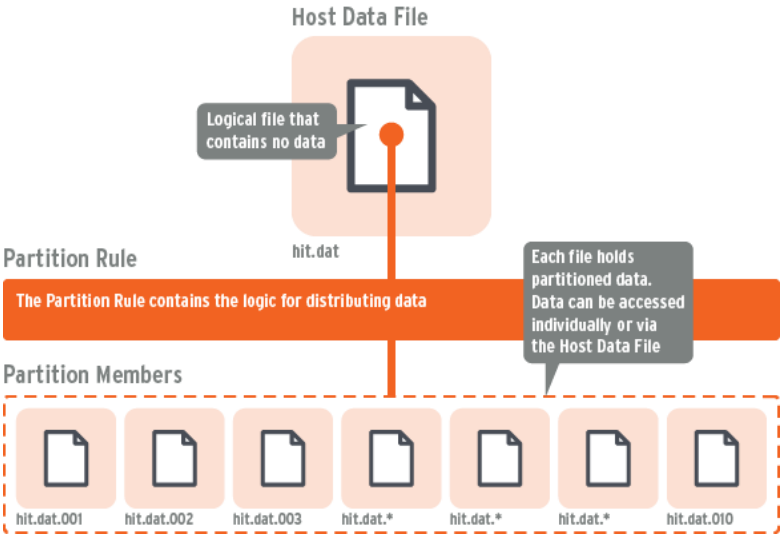
Data Compression has been implemented within the low-level record I/O. Recent challenges of larger file sizes, downsizing needs and migration compatibility from other systems already supporting data compression (as well as other customer requirements) drove our efforts on this core I/O enhancement.

Data volumes are exploding, and, as a result, database file sizes are proportionately increasing. Managing large files remains an important consideration. Compressing data is a valuable technique to reduce this data storage challenge. As low-level data records are read from or written to disk, just before they are passed to the operating system's file system, c-treeACE now intervenes and compresses before writing and un-compresses after reading each data record. In some cases we've observed compression ratios of 80:1 and more.



For more information, see the **Data Compression** (page 24) section later in this document.

Partitioned Files - Major Performance Improvements



c-treeACE V10 represents a major engineering milestone related to Partition File support. The ability to define criteria rules that direct data I/O to separate physical data files (partitions) is a powerful feature.

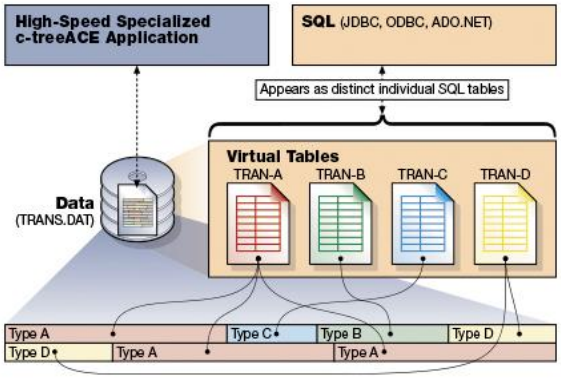
An extensive focus on the speed and performance of partitioned files highlighted numerous areas for increased efficiency and performance gains. Improvements in internal indexing strategies resulted in significant performance overall. A number of other issues were also addressed.

Global unique indices, range requests, alter table support, Windows/Unix time-stamp support, and partition administration have all been greatly improved. These enhancements are most evident when using c-treeACE SQL over a partitioned file.

For those new to c-treeACE, a partitioned file logically appears to be one file (or more accurately, one data file and its associated index files); however, it is actually comprised of a set of files whose contents are partitioned by the partition key. Both data and index files are partitioned. This permits data with a defined range of values for the partition key to be rapidly purged or archived rather than a record-by-record delete within this range. For example, a single date range can be dropped with a single call.

For more information, see **Partitioned Files** (page 27).

Multiple Record Formats



Introducing a unique new feature you will not find with any other database vendor. We have added the ability to handle data files containing multiple proprietary data record formats and map them into a relational representation. This provides the ability to operate on that data from our SQL layer, opening your unique file formats to industry standard SQL.

Many users have taken advantage of the flexibility of c-tree's ISAM or Low-Level record-oriented interface and defined their own data record layouts. In many cases, these layouts do not conform to a relational model required by SQL. Proprietary data types, multiple record types based on "record type" flags, are common in many specialized c-tree applications. Now, with our support to handle this type of non-relational data, we can convert data formats at runtime and map non-relational data into a relational model "on-the-fly" and access this data via SQL. You retain the original application's ability to operate on the data as your actual data is not modified.

We have also implemented this technology in our COBOL product. Many COBOL applications have this type of record-oriented (ISAM) data and require access to SQL without exporting / importing to an alternate SQL database. c-treeACE SQL can operate over this native original data.

For more information, see **Multiple Record Formats - FairCom DB API Virtual Tables** (page 34).

Data Filter Stacks

A significant extension to existing data record filter support takes filtering to a new level. Now you can "stack" data record filters over the same data file. Layering conditional expressions at run-time provides extensive power.

Often in an application's logic, data read requirements might be located in different areas of your code and necessitate different criteria. You might establish a data filter in one location and need additional filtering elsewhere. Say you establish a filter based on user at logon, then later in the application, a date is introduced. With the ability to "stack" your data file filters, you can add the required filter when you need it.

Filters provide an efficient way to mask what you see in a file:

- Powerful regular expression support for filters
- Stack multiple data filters on the same file
- Conditional expressions: reference any section of the record (offset, length)
- Create a filter when no schema is defined in the file resource (no DODA)
- Filter user-defined callbacks
- Permanent System-Wide File Filters

- Fast low-level (server-side) compare routines

For more information, see **Data Filters** (page 38).

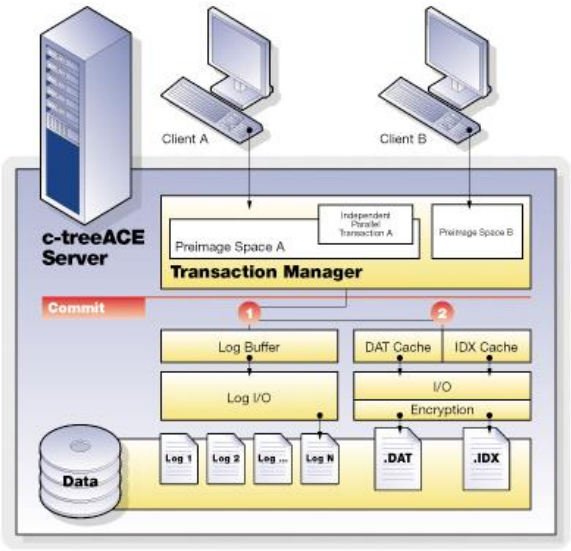
New Result Sets

Our “Result Sets” feature (FairCom DB API layer) is a smarter and more convenient way to set filters. The “result set” handle is allocated for a specific table handle and it is then possible to add one or more criteria. Once the “result set” has all the criteria added, it can be turned on and off for any record handle that is allocated for the same table handle that already owns the result set. See the section titled **FairCom DB API C and C++** (page 91).

Transaction Processing

Independent Parallel Transactions

The ability was added to perform a unique type of transaction. This special case feature is an extension to traditional database transaction processing. After you “begin” a transaction (Transaction A), and perform operations, you may now “begin” what we call an Independent Parallel Transaction (Transaction B). When this secondary transaction is committed or aborted, only the operations performed within it are affected. Operations performed within the primary transaction (Transaction A) retain their traditional scope.



This allows you to perform a transaction that is not necessarily part of the atomicity requirements of the primary transaction. If the primary transaction is aborted, the Independent Parallel Transaction will not be reverted. One could also abort the secondary transaction as a clever technique to clean up and remove temp files created by it.

For more information, see **Independent Parallel Transactions** (page 43).

Automatic Recovery

Advances in Automatic Recovery ensuring integrity of your data include:

- **Improved Performance** - A number of improvements to the performance of Automatic Recovery are included in this release.
- **Redirecting filenames during automatic recovery** - c-treeACE now supports a file originating in one directory structure to be repositioned into another directory location during automatic recovery.
- **Automatic Recovery Details** - Automatic Recovery is used to bring transaction controlled c-tree data and index files back to a consistent state in the event of an unplanned server

outage. The recovery process occurs over several phases. Various details of these phases can now be observed with the RECOVER_DETAILS YES server configuration option.

- **Additional Recovery Logging** - c-treeACE recovery has been further modified to enhance the logging of details with additional messages specifying phase and progress during the automatic recovery process. Index recomposition can frequently be a phase that takes the longest amount of recovery time. As it is useful to know the proceedings of this phase, specific messages are now output describing the status of this phase, and the index member involved as it occurs.

For more information, see **Auto Recovery** (page 44).

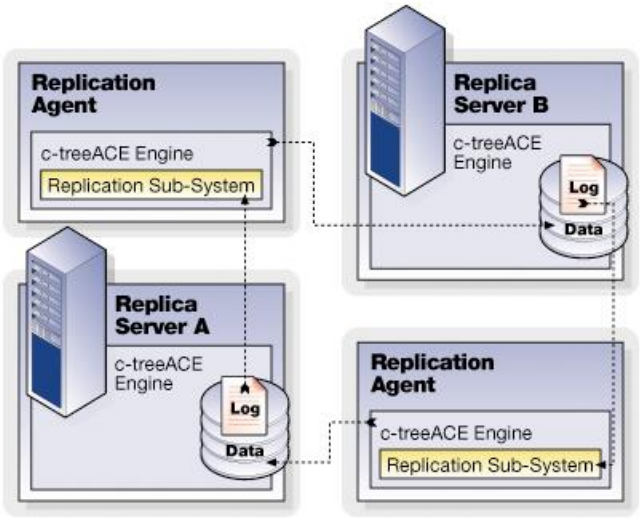
Additional enhancements to Transaction Processing include:

- Global transaction number threshold warnings
- Increased number of SAVEPOINTS
- Prevent excessive accumulation of transaction logs
- Avoid overlapping Checkpoints
- Deferred Transaction File Number Assignment
- And many more...

Replication

High Speed Availability

The c-treeACE database engine provides a flexible set of facilities for data replication between servers. This replication mechanism is implemented through transactional replication, where replication is based upon the transaction logs maintained by a c-treeACE database engine. Changes are read from the transaction logs of one c-treeACE database engine and applied to another.



Many models of replication exist to solve various data access, performance, and business continuity strategies including, and certainly not limited to:

- A single source/target model for failover and backup
- The active-active model for performance, load balancing and redundancy
- A single source with multiple targets model for local performance with a central repository

Several c-treeACE replication utilities are provided to monitor, troubleshoot and benchmark the replication process. Deployment of some replication features requires additional licensing. We encourage you to contact us and permit our team to help with your replication requirements.

Improvements in the Replication Infrastructure

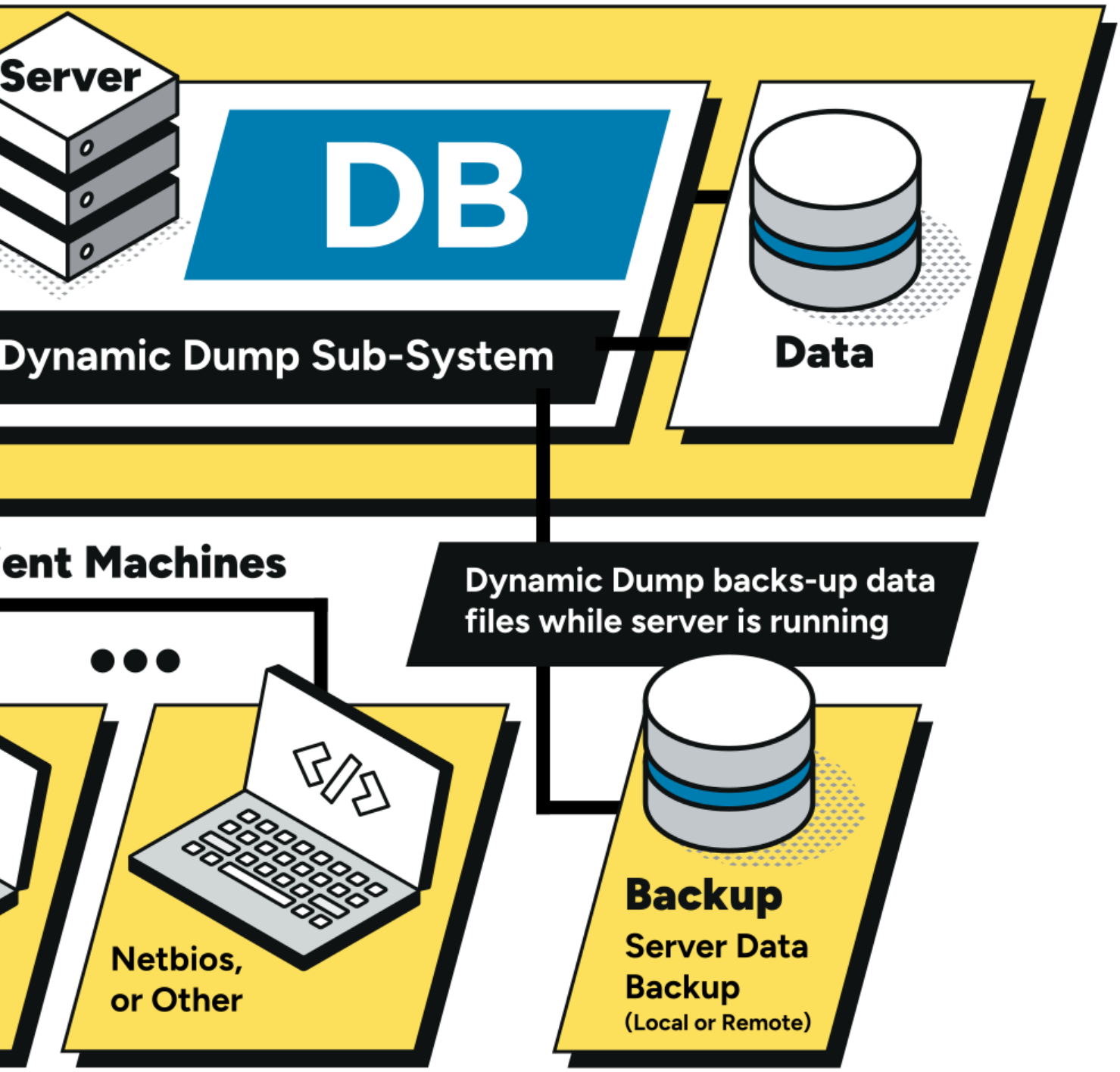
Replication is an important technology. Current c-treeACE replication support is implemented at the low-level transactional layer. Designed as a high-speed replication engine, it is intended to support high availability, redundant resources, improved reliability, and add fault-tolerance. We offer a solid infrastructure and provide engineering expertise to guide customers in implementing this technology. Significant advancements complement this release, including:

- Replication Agent supports mapping data files on a source server to different filenames on a target server
- Support for running the Replication Agent as a Windows service and writing to the Windows event log
- Added pause/resume options to the Replication Agent
- Persisted replication states allow the Replication Agent to maintain the last committed transaction applied to the target
- Support replication of compressed data
- Support replication of partitioned files
- Improved Replication Agent exception handling with failed file opens
- Options added to specify maximum attempts to lock a record and how long to sleep between attempts
- Support for replicating serial segment (*SRLSEG*) data and IDENTITY fields

For more information, see **Replication** (page 50).

Database Backups

Dynamic Dump Changes



Additional improvements to c-treeACE backup technology include:

- **Mask Routine Dynamic Dump Messages in Status Log**. Normally a dynamic dump writes the names of all the files it backs up to the c-treeACE Server status log. A new option can be specified to suppress the logging of file names backed up by a dynamic dump operation reducing the amount of logged information. See **Mask Routine Dynamic Dump Messages in CTSTATUS.FCS** (page 60, **/doc/faircom-database-backup-guide/database-backup-mask-ctstatus.htm**).
- **Ability to Kill Dynamic Dump Threads**. It is now possible to terminate a dynamic dump thread waiting for a scheduled dump time. See **Improvements in Backup/Restore** (page 59).
- **Enhanced Logging of Dynamic Dump Status Messages**. The dynamic dump process can now log the names of the files it backs up to the SYSLOG file. See **Enhanced Logging of Dynamic Dump Status Messages** (page 60).
- **Examine Dump Files Utility** now displays contents for files over 4GB. See **ctidmp Examine Dump Files Utility** (page 121).
- **Segmented File** issues related to Dynamic Dumps have been addressed.

Volume Shadow Service (VSS)

VSS is a Microsoft technology built into the Windows operating system that allows applications to access a “point-in-time” snapshot of a logical drive.

This service allows taking manual or automatic backup copies or snapshots of data. Snapshots have two primary purposes: they allow the creation of consistent backups of a volume, ensuring that the contents cannot change while the backup is being made; and they avoid problems with file locking.

By creating a read-only copy of the volume, backup programs are able to access every file without interfering with other programs writing to those same files.

c-treeACE now supports VSS through its VSS writer which controls how c-treeACE data is set to a consistent state at the beginning of a VSS operation and maintains that consistency throughout the process.

The VSS writer has been added as an integral component of the c-treeACE Server for Windows. This component is supplied as a Windows dynamic link library (c-treeACEVSSWriter.dll) and can be optionally loaded by the c-treeACE Server at startup.

For more information, see **Volume Shadow Copy Service (VSS) Integration** (page 55).

Segmented Files

Segmented Files allow a single c-treeACE data file to span multiple hard disk volumes. Several internal issues were addressed related to non-transaction processed files, un-buffered I/O, and compacting file segments. For more information, see **Segmented Files** (page 63).

2.3 Core Enhancements

Quiesce Server Activity

Additional improvements benefit the powerful ability to suspend (quiet or quiesce) c-treeACE Server operations and later re-enable them. This allows administrators to perform maintenance or other on-demand activities without having to stop an application. Users are temporarily held back from operations. Files can be readily accessed for backup, especially useful for hardware-based disk snapshot utilities. See **Quiesce** (page 74).

Shared Memory for UNIX

The c-treeACE Server and c-treeACE clients now support a shared-memory communication protocol on Unix systems. When c-treeACE detects a request to connect from the same machine as the client, it first attempts to connect using shared memory instead of TCP/IP. The shared-memory communication protocol on Unix systems has excellent performance results on most systems over TCP/IP. See **Inter-Process Communications - Shared Memory for Unix** (page 65).

Broadened Auto-Numbering Support

Sequential numbering is a requirement for many applications. Accounting data is a particularly heavy application usage with respect to invoices, quotations, and other receivables and payables documentation. c-treeACE offers two new methods of automated numbering.

Consider global operations against multiple servers that require a single numbering scheme. This requires a persisted pool, and indeed, many c-treeACE applications have implemented their own unique solutions. c-treeACE V10 introduces a generic method to create, maintain, and access multiple sources of sequential numbers through a new persisted **Sequence** (page 71) feature.

An auto-incremented IDENTITY field has also been added. Specify the base and increment for a numeric field and each time a new c-treeACE record is added this field is updated. For more information, see **Auto-Numbering Support** (page 70).

2.4 Security

c-treeACE services mission-critical applications in such sensitive industries as banking, finance, and health care. The increasing demand to provide users with highly secured access to systems and data continues to impact developers in these and many other industries. Providing the necessary security required to safeguard mission-critical data is crucial.

In our ongoing reviews we have implemented additional defenses for securely storing and transmitting your data as follows:

- Advanced encryption master key length has been increased from 128 to 256 bits. c-treeACE's advanced encryption uses a master

encryption key to encrypt the following items using the AES cipher: a) the security resource in c-treeACE data and index files b) the encryption key in the transaction log file header. We also use a 256-bit key to encrypt the transaction logs and the *FAIRCOM.FCS* user information file.

- Added a master encryption key store. Support was implemented for specifying a file from which to read the master encryption key.
- Tighter security restrictions prevent a member of the ADMIN group from changing the password of the super administrator account. Prior to this change, c-treeACE allowed ADMIN group users to change the password for any ADMIN user account, including the super ADMIN account.

For more information, see **Security "Lockdown"** (page 79).

2.5 More Features

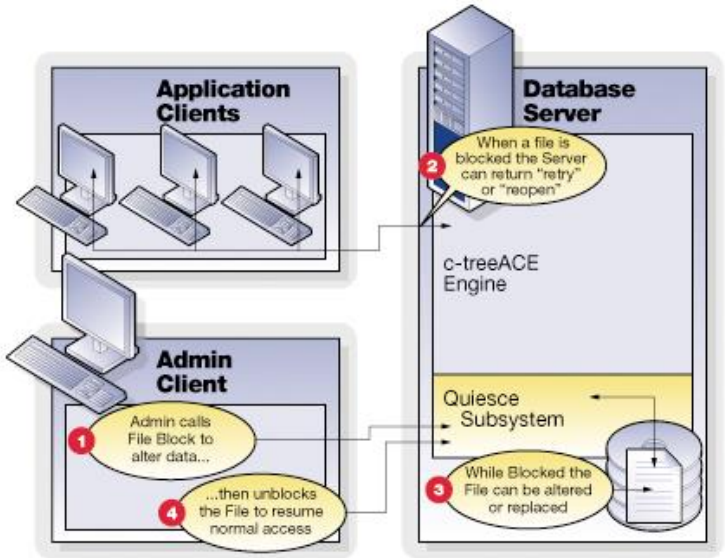
Thread Impersonate - This advanced feature supports a connection impersonating another connection. The c-treeACE Server maintains a separate thread for each connected client. Each server thread has its own internal "Owner ID" to maintain the context information related to that user/connection. A new API can be called to evoke a connection to impersonate another connection.

c-treeACE listens on two separate communications ports: one for SQL, and another for ISAM connections. When a SQL client connects, it has its own server thread/context. When an ISAM client connects, it has its own server thread/context. Customers requested both the SQL and ISAM operations to be coordinated under a single database transaction which was not previously possible within the same transaction context.

A user may now connect a SQL client and an ISAM client and then indicate to the server that the ISAM client wants to impersonate the SQL client. On the server-side, both connections use the same "Owner ID" allowing the server to view them as the same thread/context. Sophisticated users can now have SQL and ISAM code intermingled within the same code/database transaction. This is a great technique to gain performance advantages when a well-crafted ISAM call is preferable to SQL query overhead.

Thread Impersonate is an advanced feature that requires careful implementation. For more information, see **Thread Impersonate - Connection Impersonating Another Connection** (page 72).

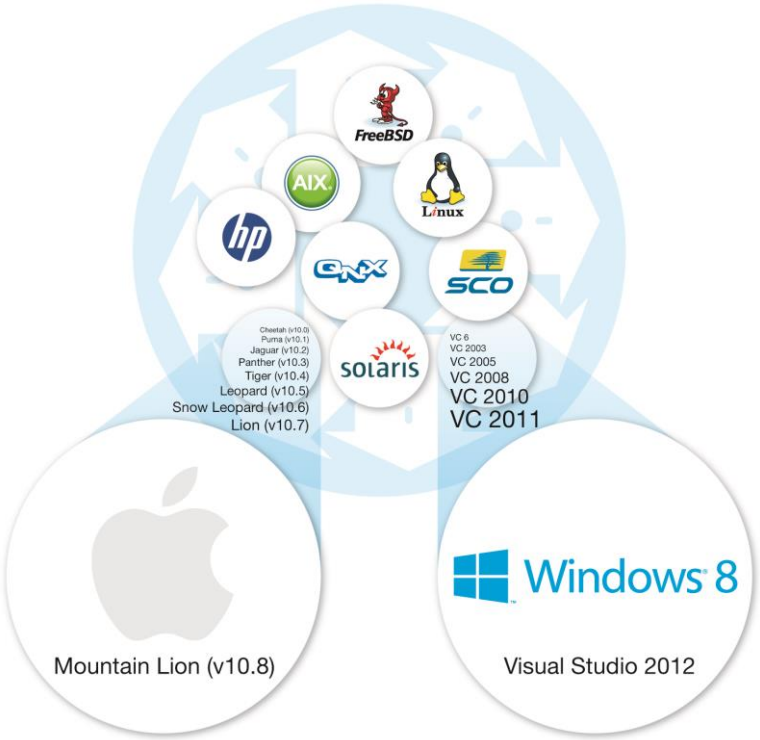
Memory Files - Exceptional in-memory file and index support offers blazing I/O speeds. We have improved this performance even more in this release. Concurrency of operations on memory files is improved when adding records. Also, on 64-bit systems with memory address values over 4GB, an alternative hash function has been implemented that improves overall performance. See **Memory Files** (page 63).



Distinct Key Counts for Duplicate Index - To optimize SQL queries, a more precise estimation of the duplicate index selectivity was required, which implies knowing how many distinct keys there are in an index allowing duplicates. Counting distinct partial key values on indices improves the index selectivity calculations and results in overall improved performance of c-treeACE SQL. See **Distinct Key Counts for Duplicate Index** (page 62).

Diagnostics and Monitoring Improvements - Including disk-full conditions, memory allocation tracking, run-time control over low-level file I/O diagnostics, improved internal statistics available from the SnapShot API call and ctstat utility.

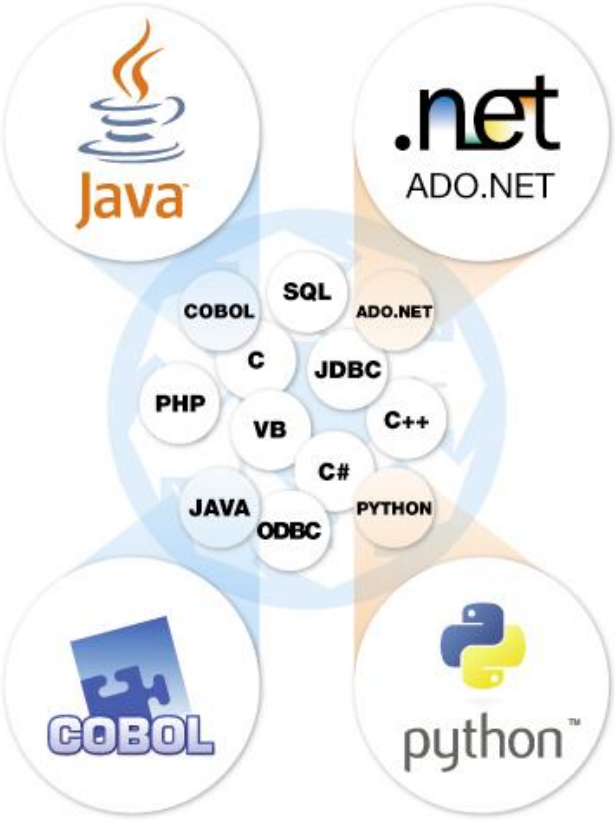
2.6 Platform Support



Supported platforms include:

- Windows Vista through Windows 8; Visual Studio 6 & 2003 through 2012
- HP Unix (HP-UX)
- IBM Unix (AIX)
- Solaris
- QNX
- SCO
- Linux
- FreeBSD
- Apple v10.0 through v10.8

2.7 Interface Technology



ISAM

Indexed Sequential Access Method (ISAM) record access technology has been proven over many years. It remains a technology of choice for high-performance applications due to its high throughput and low cost of maintenance.

SQL

Structured Query Language (SQL) refers to any database management technology that utilizes this ANSI/ISO industry-standard access method.

Best of Both Worlds

FairCom's unique competitive advantage is in providing both ISAM and SQL access to data concurrently. By providing interfaces directly into the ISAM storage engine, a developer can avoid the performance overhead of SQL processing. This unparalleled flexibility allows application developers to choose the degree of relational control they require to efficiently implement their requirements. The end result for you is the best possible performance with the richest feature set.

For information about enhancements in the many interfaces supported by c-treeACE, see **Interface Developments** (page 85) and **c-treeACE SQL Advances** (page 97).

c-treeACE SQL

c-treeACE SQL provides a high-performance SQL interface into the proven core of the c-treeACE Server. Tailored for high volume production environments, the c-treeACE SQL Server includes optimizations such as sophisticated query rewrite techniques to improve nested query performance and join-order optimization to improve performance of queries joining many tables. c-treeACE SQL extensively caches and buffers information for maximum transaction and query throughput.

Because c-treeACE SQL is built on the same core technology as the c-treeACE Server, you get all of the performance and features that distinguish c-treeACE with the additional benefits of a functionally-complete SQL interface compatible with SQL-92, ODBC 3.0, and JDBC 3.0 standards.

c-treeACE SQL provides embedded SQL and interactive SQL utilities, as well as ODBC, JDBC, ADO.NET, PHP, and Python drivers. c-treeACE SQL also supports stored procedures, triggers, and user-defined functions utilizing the powerful and extensible Java language.

FairCom's c-treeACE SQL supports most of the major platforms including Windows, Linux, Mac OS X, HP-UX, Solaris, and AIX. Additional ports to other operating systems are constantly ongoing, so please contact FairCom for the latest status of your operating system of choice.

Notable SQL Features

- LONG character and binary fields (up to 2GB)
- Stored Procedures, Triggers, and User-Defined Functions
- Recursive views
- GRANT and REVOKE support at the column, table, and view level
- Utilities for importing data
- Easily integrated into Visual Studio and ADO.NET Entity Framework
- Graphical tools for database creation and management

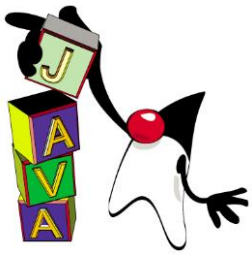
For more information, see **c-treeACE SQL Advances** (page 97).

2.8 New Java Development

Java is a recognized industry-standard, cross-platform development environment designed to meet the challenges of application development in the context of heterogeneous, network-wide distributed environments.

FairCom has several exciting new projects under development involving Java:

JTDB



The FairCom DB API Java, JTDB, native Java interface utilizing the FairCom DB API paradigm. This is a record oriented c-treeACE interface providing advanced flexibility to the Java developer. This record-oriented c-treeACE interface utilizes the simplified concepts of sessions, databases, and tables in addition to the standard concepts of records, fields, indices, and segments. It allows Java applications access to data with the same high throughput rates previously only available to C, C++, and .NET applications.

JEE

The FairCom implementation of a Resource Adapter for JEE Application Servers (Jboss, Websphere, Weblogic, Tomcat). This JEE adapter publishes c-treeACE ISAM methods making them available for applications using Enterprise JavaBeans (EJB) deployed on an application server framework. It allows these applications to view c-treeACE data as a registered resource. It is used for enterprise-class, distributed applications integrating multiple environments, such as mainframes and open systems, requiring fast ISAM access to c-treeACE data.

JPA

The Java Persistence API (JPA) is a Java programming language framework managing relational data in Java programs. This framework allows Java developers to abstract their data interactions without writing specific data I/O commands of any kind. It allows applications to be independent of the platform and database and creates a fully abstract, 100% object-oriented implementation of that infrastructure. Common JPA implementations, such as Hibernate, map data interactions to SQL commands and use an RDBMS to persist the data. FairCom offers a unique implementation: Instead of mapping to SQL, we map to our ISAM interface, which uses fewer resources and improves performance. The FairCom JPA can replace Hibernate and the RDBMS in any application that has been developed using this architecture with minimal changes in the Java code. It is worth noting that FairCom also supports a generic dialect of Hibernate and other SQL JPA implementations through our c-treeACE SQL/c-treeACE JDBC interface.

Java Tools

FairCom now offers Java versions of its tools, providing cross-platform compatibility. See the **GUI Tools** (page 102) section of this document for more about the rich set of tools FairCom offers.

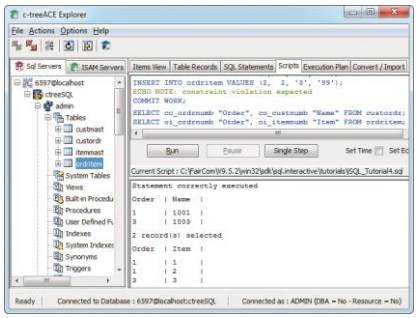
If you are interested in putting any of these technologies to work for you, please contact FairCom Sales.

2.9 Updated & Improved Tools

Java Tools for Every Platform

- Same look and functionality as c-treeACE .NET Tools
- Cross platform—Windows, Mac, Solaris, AIX, Linux & Unix
- Combined ISAM & SQL views

From the lowest-level ISAM data file diagnosis to complex SQL queries, c-treeACE has a tool for the job.



c-treeACE has extended its tools offering to include alternative platforms. By authoring the tools in Java, many new platforms can be utilized, including Mac and Linux. These Java-based tools have been written to completely match the look and functionality of the original .NET-based Windows tools, with many new feature additions.

- ISAM search by key value
- Open external tables
- XML data output
- And many more...

Development

Quickly design and modify tables and indexes. ISAM Explorer provides additional control. Execute build scripts from SQL.

- c-treeACE ISAM Explorer
- c-treeACE SQL Explorer
- c-treeACE Load Test

Administration

Easily manage and configure user security. Analyze status logs and view current system performance.

- c-treeACE Security Administrator
- c-treeACE SQL Query Builder
- c-treeACE Status Log Analyzer
- c-treeACE Performance Monitor

Monitoring

Proactively monitor your system with graphical ease. Select only the statistics you require. Over 250 metrics available to monitor.

- c-treeACE Gauges
- c-treeACE Monitor
- c-treeACE Performance Monitor

...PLUS

.NET Tools for Windows

- Native .NET
- 32 and 64-bit support

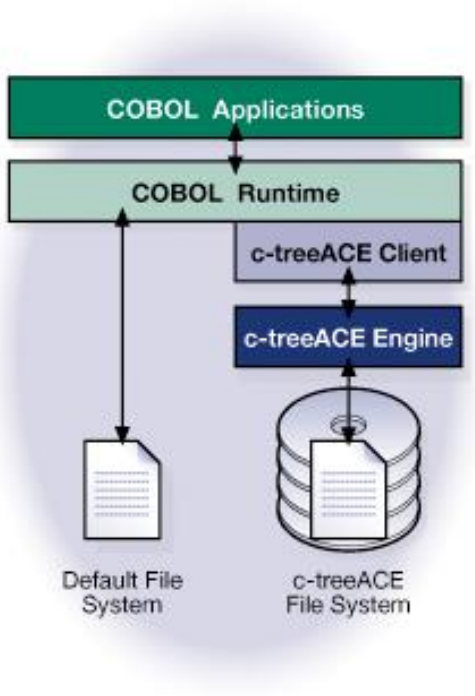
Command Line Flexibility

Don't forget! Over 30 scriptable command line tools are included making nearly any administrative task possible. Utility source code is also included allowing you to custom tailor any tool to your exact needs.

For more information, see **GUI Tools** (page 102) and **Command-Line Utilities** (page 109).

2.10 FairCom RTG COBOL

FairCom's FairCom RTG COBOL is a sophisticated file system designed as a replacement for the default file systems available with COBOL.



The basic functionality provided by many of the default COBOL file systems is in some cases unable to satisfy modern application needs. Often developers are forced to migrate to complex and expensive relational systems. These systems may place a heavy impact on the COBOL application both in terms of performance and code modifications.

The FairCom solution maintains the simplicity of the standard COBOL file system interface while offering COBOL developers high-end database technology and additional features such as transaction processing and a client/server model. Existing applications integrate with FairCom RTG COBOL without the need to recompile. In fact, FairCom RTG COBOL has been implemented to fit ACUCOBOL-GT's and Micro Focus ExtFH architectures, providing an easy replacement to the default file system. Developers have the flexibility to choose which file system to use on a file-by-file basis.

In addition, FairCom RTG COBOL transparently allows access to data from your COBOL application through SQL and other FairCom interfaces such as ODBC, JDBC, and ADO .NET drivers.

FairCom RTG COBOL is sold separately. For more information, visit the COBOL page at **www.faircom.com** (**http://www.faircom.com/**).

3. IMPORTANT: New Licensing & Activation

c-treeACE V10 features an improved, more user-friendly activation process. The new licensing and activation process is described in this section. You will need to use the procedures in this section to activate your c-treeACE V10 release.

3.1 V10 Licensing Mechanism

Note: You must recompile clients to access the V10 release of c-treeACE.

See also **V10 Client/Server Compatiblity** (page 214) for important information about compatibility between releases.

If you have questions, contact your nearest FairCom Support team.

3.2 Limit Concurrent Logons by User Name/Group

c-treeACE Server now supports limiting users to a specified number of concurrent logons based on their user name and/or group membership. By default, these limits are zero, meaning no limit.

The **ctadmn** utility's menus support two new options to administer this capability: User Operations menu > Change User Logon Limit and Group Operations menu > Change Group Logon Limit.

The **SECURITY()** API function supports two new modes: SEC_CHANGE_USER_LOGLMT and SEC_CHANGE_GROUP_LOGLMT.

See **User and Group Logon Limits** (page 83) in the Security section.

4. File Management Milestones

This release has seen many advances in the area of file management, from low-level data I/O to operational areas such as backup/restore and replication. This section lists those changes.

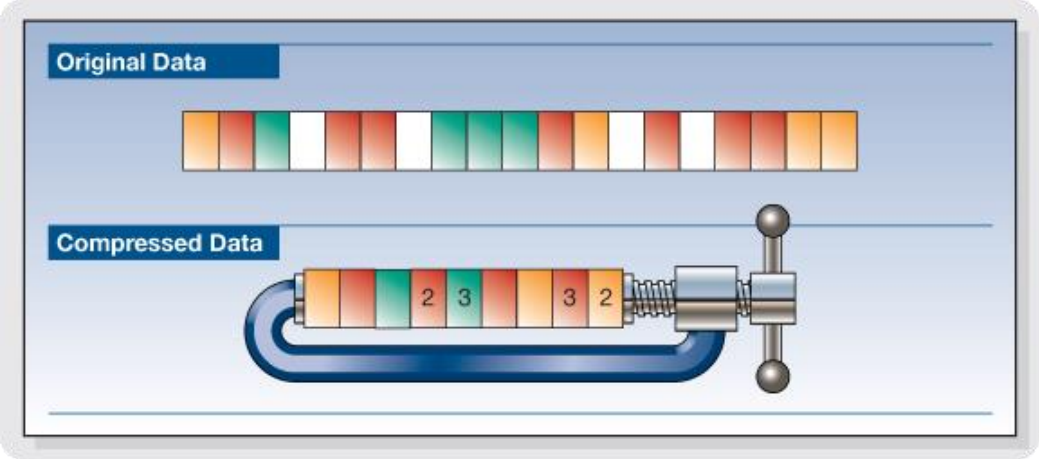
4.1 Data Compression

Data volumes are exploding, and, as a result, database file sizes are proportionately increasing. Managing large files remains an important parameter in sizing disk arrays, which can be quite expensive for high availability systems. Simply moving large data sets for archiving and backup purposes becomes a challenging task. Compressing data is a valuable technique to reduce this data storage challenge. By directly reducing the data record size, file sizes can be substantially reduced.

To enable support for additional file modes c-treeACE has implemented augmented variable-length records. The first supported augmented feature being data record compression.

c-treeACE Data Compression

c-treeACE now supports Data Compression. Recent challenges of larger HUGE files, downsizing needs, and migration compatibility from other systems that support data compression, as well as specific customer requirements resulted in this core database necessity.



As low-level data records are written to and read from disk, c-treeACE now intervenes just before they are passed to the operating system's file system, and will “compress” before writing and “un-compress” after reading each data record.

The default compression algorithm comes from the standard zlib library, written by Jean-Loup Gailly and Mark Adler and is an abstraction of the DEFLATE compression algorithm used in their gzip file compression program.

c-treeACE also supports a proprietary run-length encoding (*RLE*) option; yet perhaps even more important is the ability for the user to implement a Call-Back where you can define your own compression technique.

Data Record Compression

As different files and applications may require various types of compression, a c-treeACE resource (*CMPRECRES*) has been defined and is embedded into files supporting data record compression. This resource contains the compression type, version, custom parameters, and the DLL name (if built-in compression is not used). The calls to the compression routines are handled by function pointers that are automatically initialized for the built-in routines and DLLs.

The compression resource structure supports optional parameters to fine tune the compression operations. For example, zlib has five parameters that control internal processing. The compression structure uses a *pVOID* pointer and a length parameter to permit different compression DLLs to support different parameter sets. **ctSETCOMPRESS()** (page 150) allows an application to set these parameters.

Server Configuration

Default c-treeACE compression can be specified with the following configuration keywords:

```
CMPREC_TYPE         < "ZLIB" | "USER" >
CMPREC_VERSION              <a number >= 1>
CMPREC_DLL          <name of DLL>
```

These keywords should be entered in the configuration file in the order shown, and a DLL name is required for `CMPREC_TYPE` of USER.

A new COMPRESS_FILE keyword can be used to enable compression in files whose names match specified file names (including wildcards). See **COMPRESS_FILE** (page 26).

Compressed Files in FairCom DB API

A new FairCom DB API file create mode, *CTCREATE_COMPRESS*, has been introduced to enable compressed record support in FairCom DB API. When this mode is used, FairCom DB API automatically creates the file as variable length.

Notes

- Some files will compress better than others (text data vs. binary data for instance). Initial testing revealed up to an 80:1 compression ratio in some cases.
- At this time, only zlib file compression is available, however, support has been included for multiple compression routines in the future.
- This feature is available only for ISAM files fully maintained by ISAM updates. Low-level c-treeACE functions will return an error when used on files with the compression attribute enabled.

COMPRESS_FILE

```
COMPRESS_FILE  <filename>
```

Forces c-treeACE to create data files whose names match the specified name with data compression enabled. The file name may include wildcard characters (see **c-treeACE Standard Wildcards**).

A file whose name matches the keyword is created as a variable-length data file even if the create option specifies that it is a fixed-length data file. Such a file is still restricted to containing records that have the defined fixed-record length.

The logic has been changed so that the `COMPRESS_FILE` keyword does not enable data compression for a data file that is created with resource support disabled. This prevents a rebuild from failing with **error 484** (could not open sort work file).

See Also

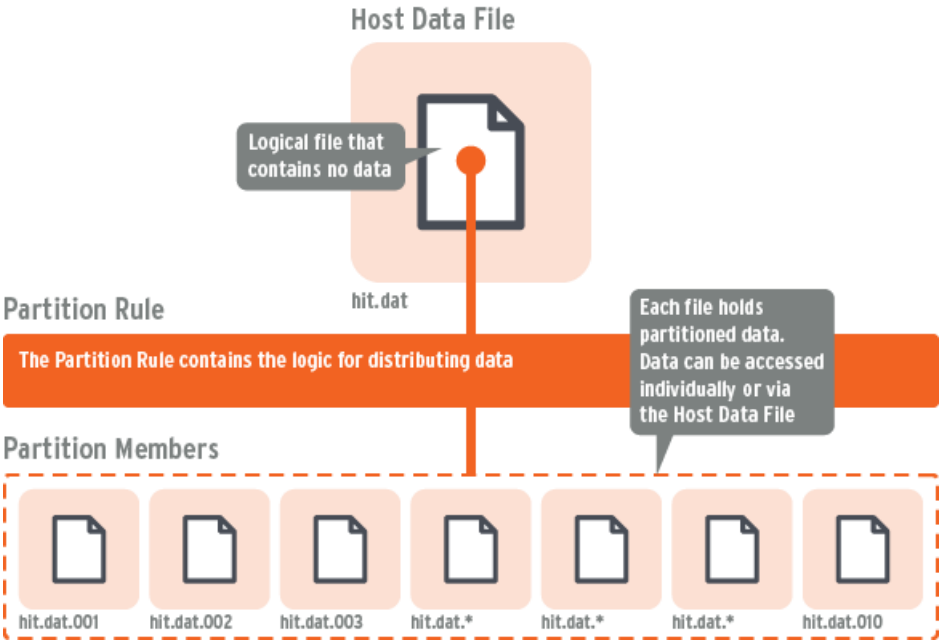
- **CMPREC_TYPE in the** *c-tree Server Administrator's Guide***.**

4.2 Advances in Partitioned Files

The release represents a major engineering milestone related to our Partition File support. The ability to define criteria/rules that direct data I/O to separate physical data files (partitions) is a very powerful feature.

An extensive focus on performance has driven most of our efforts. Improvements in our internal index strategies resulted in significant performance overall.

Global unique indexes, range requests, alter table support, Windows/Unix time-stamp support, and partition administration have all been improved. These improvements are most evident while using our SQL layer over a partitioned file.



Partitioned Files Available via c-treeACE SQL

Partitioned files are available directly through c-treeACE SQL. As partitioned files rely on a key value to partition the data, the option is part of the CREATE INDEX statement. By including the STORAGE_ATTRIBUTES clause with the PARTITION option, the file will be rebuilt to enable partitioned support. All other operations on the file will continue as usual.

Note: Partitioned file support requires a custom build of c-treeACE SQL with the partition rule compiled from the *ctpart.c* module.

c-treeACE SQL CREATE INDEX Syntax

```
CREATE [ UNIQUE ] INDEX index_name
         ON table_name
       ( {column_name [ASC | DESC]} [, ...])
       [ STORAGE_ATTRIBUTES 'attributes' ];
```

New Argument:

STORAGE_ATTRIBUTES 'attributes'

A quoted string that specifies index attributes that are specific to c-treeACE SQL indexes. c-treeACE SQL interprets the following attribute strings:

- 'PARTITION' - Creates the index as the partition index to the table enabling multiple table partitions. This option can be specified after a table has been created, and the table will be rebuilt according to the partition rule in effect.

Optimized c-treeACE SQL Partitioned File Queries

A detailed analysis of how partitioned files were opened and queried by various SQL constructs was taken. Many enhancements were identified that could greatly improve performance when multiple physical files are taken into consideration:

- **Estimation of key values** - c-treeACE SQL requires an estimation of key values as part of the query optimization phase. It was discovered that this phase of query execution frequently consumed the largest amount of time when working with large numbers of partitioned data files. It was found that the calling of key estimation routines opened large numbers of files to obtain the key estimate. To better optimize this phase, a sampling technique is now performed on a much smaller subset of partitions to reduce time spent in this critical phase. The partitions sampled are the first and last partitions that ordinarily would have been used, and one or more in the “middle” of the remaining active (or covering) partitions.

  c-treeACE SQL defaults to three samplings. The following configuration keywords change this behavior:

  - `PARTITION_ESTIMATE_LIMIT <limit>` increases this limit to a desired value. A negative value resorts to the previous behavior of reading from each active partition (or covering partition).
  - `PARTITION_ESTIMATE_LIMIT <limit>%` increases this limit as a percentage of eligible partitions.
- **Active number of key values** - An enhanced ability to return the active number of key values without having to examine each active partition member. All necessary information is now stored in the host Global Unique Index (GUIx).
- **Query logic modifications** - Query logic was modified to check for empty covered ranges, and also to check for unexpected missing partitions in middle partitions that are sampled.
- **Range search** - When a range search is performed on a unique index and the range criteria specify an equality match on all segments of the key, a direct equal key function is now called rather than a key range function. For a partitioned file global unique index that does not cover the partition key, this greatly improves performance when many active partitions exist as the equal key call can use the global unique partition host index to find the partition that contains the key value directly, avoiding costly searches through multiple partitions.

- **Improved hashing** - An improved hashing mechanism for determining if a given file is already open. For large numbers of open files (such as when partitioned files are in use) this substantially reduces initial open times by reducing search times.

ALTER TABLE Add and Drop Columns Supported for Partitioned Files

The ability to add and drop columns for Partitioned Files via an ALTER TABLE (either via SQL or FairCom DB API) has been added. Previously an invalid argument error was returned (**CTDBRET_INVARG**) when attempting this operation. For very large data sets this could take time, as currently, every record is visited to update based on the new schema. In addition, if indexes require a rebuild, this will require additional time.

Partitioning by Windows Date Values

The previous date partition rule was implemented for a Unix time value (epoch 1 Jan 1970) and was based on a key value type of double (float).

This revision introduces a modification to the **kprawno()** function to interpret the first 8 bytes of the partition key value as a Windows local file time timestamp (that is, the timestamp contains the number of 100-nanosecond intervals since January 1, 1601). This option is useful when an application uses the .NET *DateTime*.**ToFileTime()** method to generate the timestamp values that are used in the partition key.

Internally, this support converts Windows format to Unix format for flexibility in the partition logic. As a result, it is only enabled for Windows compiles and allows for both formats to potentially be used.

Partition by Timestamp Rule Modified to Support GMT/UTC Time

When an attempt to read data from a partitioned file on a machine whose time zone differs from the time zone on the machine that stored the data, the data was not found. This happened as the function that maps the partition key, a timestamp, to a partition number, converts the timestamp to year, month, and day using the local time zone. To provide a consistent mapping of timestamp values to year, month, and day (which determines the partition number), an additional option has been added using GMT instead of the local time zone to convert the timestamp value.

New GETFIL() Modes to Retrieve First and Last Active Partition Members

When working with partitioned files, it is useful to quickly locate the first (the "oldest" when partitioning by date) and last ("newest") partition members for administrative purposes. Two modes have been added to the **GetCtFileInfo()** function to support this ability.

- *FRSACTPRT* - returns the first active partition number for the file
- *LSTACTPRT* - returns the last active partition number for the file

If **GETFIL()** returns -1, check the value of *uerr_cod*. If *uerr_cod* is zero, the file has no active partitions. If *uerr_cod* is non-zero, an error occurred. For example, if the specified file number

does not correspond to a partition host file, **GETFIL()** returns -1 and sets *uerr_cod* to **PHST_ERR**.

Example

```
COUNT datno;
NINT  rc;
LONG  partno;

if ((partno = GETFIL(datno, FRSACTPRT)) == -1L) {
        if ((rc = uerr_cod))
                printf(
        "Error: Failed to get first active partition number: %d\n",
                        rc);
        else
                printf("The file has no active partitions.\n");
} else
        printf("first active partition: %d\n", partno);

if ((partno = GETFIL(datno, LSTACTPRT)) == -1L) {
        if ((rc = uerr_cod))
                printf(
        "Error: Failed to get last active partition number: %d\n",
                        rc);
        else
                printf("The file has no active partitions.\n");
} else
        printf("last  active partition: %d\n", partno);
```

FairCom DB API .NET Interface Support for Partition File Management

The FairCom DB API and FairCom DB API .NET interface layers have been enhanced with added functions and methods for administering and managing c-treeACE partitioned files.

FairCom DB API C API

The following functions have been added to FairCom DB API for partitioned file administration:

- **ctdbPartAdminByName()**
- **ctdbPartAdminByKey()**
- **ctdbPartAdminByNumber()**

FairCom DB API C++ API

Three methods have been added to the *CTTable* class for partitioned file administration:

- **PartAdminByName()**
- **PartAdminByKey()**
- **PartAdminByNumber()**

FairCom DB API .NET

Three methods have been added to the *CTTable* class for partitioned file administration:

- **PartAdminByName()**
- **PartAdminByKey()**
- **PartAdminByNumber()**

Descriptions for the .NET methods are in the **Function Reference** appendix in **Partitioned File Management API** (page 186).

Additional FairCom DB API .NET CTTable.PartAdminByKey() Method

The FairCom DB API .NET *CTTable*.**PartAdminByKey()** method initially had only a parameter requiring a *IntPtr* type and required marshaling of data into this type, which can be somewhat cumbersome. A new overloaded method has been added to allow a *CTRecord* type and an integer key length to be passed. **PartAdminByKey()** now internally calls *CTRecord*.**BuildTargetKey()** and passes the proper target key value to the partition administration method for ease of use.

```
COUNT PartAdminByKey(CTRecord Record, int KeyLen, CTPART_MODE_E PartMode)
```

Example

```
// allocate objects
MySession = new CTSession(SESSION_TYPE.SQL_SESSION);
MyDatabase = new CTDatabase(MySession);
MyTable = new CTTable(MyDatabase);

Console.WriteLine("\tLogon to server...");
MySession.Logon("FAIRCOMS", "ADMIN", "ADMIN");
MyDatabase.Connect("ctreeSQL");

Console.WriteLine("\tOpen table...");
MyTable.Open("custmast", OPEN_MODE.EXCLUSIVE_OPEN);

// Duplicate keys allow for 8 extra bytes
dupLen = 8;
Int64 ts = 121140092800000000;
int keyLen = 8 + dupLen;

CTRecord rec = new CTRecord(MyTable);
rec.Clear();
rec.SetDefaultIndex("custmast_idx_partition");
rec.SetFieldValue("entry_date", ts);
MyTable.PartAdminByKey(rec, keyLen, CTPART_MODE_E.PURGE);
```

FairCom DB API Methods to Retrieve Partitions

Additional methods have been added to the FairCom DB API .NET API *CTTable* class to support retrieving the first (oldest) and last (newest) partition members.

FairCom DB API .NET API

```
LONG CTTable.GetFirstPartition()
LONG CTTable.GetLastPartition()
```

These methods return the partition *rawno* value of the partition member, which can then be used to purge or otherwise administer the partition member directly. A *CTException* is thrown if an error occurs and the specific c-treeACE error code should be examined from that.

This support is rooted in the FairCom DB API API with the following additions.

FairCom DB API C API

```
LONG ctdbGetFirstPartition(CTHANDLE Handle)
LONG ctdbGetLastPartition(CTHANDLE Handle)
```

Returns the *rawno* of the first or last partition for the file if partitions exist. If returns -1, call **ctdbGetError()** to retrieve the ISAM error code. If this value is zero, the file has no active partitions. If *uerr_cod* is non-zero, an error occurred. For example, if the specified file number does not correspond to a partition host file, **ctdbGetFirstPartition()** returns -1 and **ctdbGetError()** returns error **PHST_ERR** (713). **ctdbGetError()** will return **CTDBRET_NOTACTIVE**, or **CTDBRET_NOTTABLE**, if an invalid table handle is passed in.

FairCom DB API C++ API

```
LONG CTTable::GetFirstPartition()
LONG CTTable::GetLastPartition()
```

Returns the partition *rawno* value of the partition member. A *CTException* is thrown if an error occurs.

Improved Rebuilding of Partitioned Files

Enhanced partition file rebuild is now supported via three modes:

- Call **RBLIFILX8()** for the partition host which will force the host and all partitions to be rebuilt.
- Call **RBLIFILX8()** for the partition host with *tfilno* set to *badpartIFIL* (which cannot be combined with *updateIFIL*), and, if the host is clean, it will only rebuild partitions that are not clean.
- Call **PTADMIN()** for a specified partition using *ptADMINrebuild* mode.

For the first two modes of rebuild that use **RBLIFILX8()**, the *XCREblk* argument must be included as we check the *x8mode* member of *XCREblk* for partition attributes.

Partitioned File Range Optimizations

Previously, for a partitioned file, a range request using an index that did not search in partition-index order requires that certain key operations examine all active partitions to determine the correct result. For a partitioned index within a range request, the partition logic now takes advantage of the range conditions to restrict, if possible, the number of partitions that must be examined. The result of this enhancement is an increase in performance when searching data within the partitions.

Updated Partition Admin Modes Reuse and Base

After purging a partition member, that partition number is no longer available for use, as the member is marked purged. The partition administration function **PartAdmin()**, had originally stubbed in a reuse mode, however, it was not implemented, and this mode is now available for use. The *ptADMINreuse* mode only supports reuse of a previously purged partition member.

As part of this change, partition instance numbers are introduced such that the host file can distinguish between different versions of the same partition. By “same” partition we mean partitions that contain the same range of partition key values. Reasons for having different versions of the same partition include purging a partition and then recreating the partition, or rebuilding a file partition (that could result in modified contents). The instance numbers are used in the host’s global unique index (GUIx), if any.

A GUIx contains key values from all partitions as a means of ensuring global uniqueness for a key value across all the partitions. The key value is stored in the GUIx, however, instead of storing a record location to go with the key, we stored the partition number that holds the record. This way, if we purge a partition, we do not need to find all the entries in the GUIx that correspond to the partition because if we find a duplicate conflict when trying to add a new key value, we can check if the existing key value is for a purged partition, and, if so, we can replace it by the new key value for an active partition.

The new implementation now stores not only the partition number in the GUIx, but also the instance number (that defaults to zero). Now it is not only possible to distinguish between purged and active partitions in the GUIx, we can also distinguish between a purged and recreated partition since we force them to have different instance numbers.

For non-huge files, the instance numbers are in the range of 0 to 255. Huge files use four bytes for the instance number.

The *ptADMINbase* mode behavior has also been improved such that it is possible to change the base raw partition number (that is, the lowest permitted partition number) to any desired value as long as it does not exceed any active partition. Instance numbers permit purged partitions to fall outside the new base number because we can distinguish between different versions of the partitions.

When a partition is opened, its instance number is checked against the host list of instance numbers by partition. If they don’t match the open fails with error **PNST_ERR** (927).

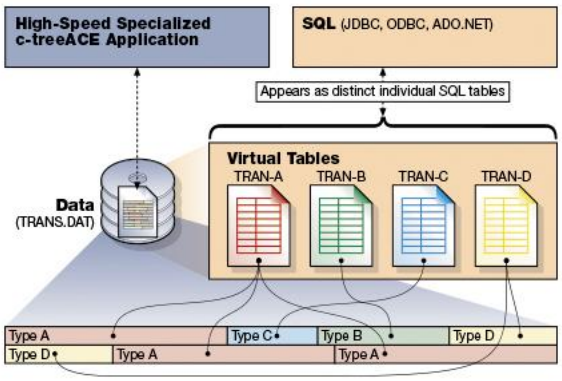
Note: The addition of instance numbers has caused the partition resource stored in the host data file to be revised, and assigned a version number 2. Prior code will not be able to open a partition file with a version 2 resource, and will fail with error **PVRN_ERR** (725).

PTADMIN() Partition Administration Purge

We now allow **PTADMIN()** purge to block access to the partition files that are being purged so it can open them in exclusive mode. This allows the call to succeed even when one connection has the partition host file opened.

4.3 Multiple Record Formats - FairCom DB API Virtual Tables

We have added the ability to handle data files that contain multiple data record formats (proprietary, non-relational) and map them into a relational representation. This gives you the ability to operate on this data from our SQL layer, opening up your data to the open standards world. This is a significant new feature that you will not find with any other database vendor.



Many users have taken advantage of the flexibility of c-treeACE's ISAM or Low-Level record-oriented interface and have defined their own data record layouts. In many cases these layouts do not conform to the relational model required by a SQL engine. Proprietary data types, multiple record types based on a "record type" flag are common in many older c-treeACE based data files. This situation is most likely in prior c-treeACE applications that are later ported to c-treeACE SQL with minimal application changes.

With support for handling this type of data, c-treeACE can convert data formats at runtime and map non-relational data into a relational model "on-the-fly" to access this data via c-treeACE SQL.

Much as many existing c-treeACE applications have these types of mixed data records, many COBOL applications also utilize this type of multi-record type data and have the need to access it via SQL. Exporting and importing the data into an alternate SQL database is messy and time consuming. As such, we have implemented this multiple record format technology in our FairCom RTG COBOL product, bringing advanced SQL access to your legacy COBOL data.

Introduction to Virtual Tables

As a proven high-performance database technology, c-treeACE has allowed application developers to craft their data handling needs precisely for their application needs for many years. Using only the necessary features required for a specific application task allows for very high-performance applications. Relational aspects of the database were maintained by the application greatly simplifying the persisted information required. As database technology has evolved, relations over the data are becoming more common and complex, and are now generally assumed to be persisted with the data. SQL is an excellent example of this, where tables are organized into databases, and columns comprise a data row within a table.

A common technique in many early c-treeACE applications was to combine multiple types of records into a single physical data file. With c-treeACE's direct record access and ISAM technology, this was very easy to implement and maintain. Many of these applications have been very successful for many years and are maintained yet today. As a successful application, developers are looking at adding more complex features and interoperability over their data with new technologies such as web access and Windows .NET support. This has proven to be a challenge when the data is not of a consistent schema due to multiple record types in a single table.

With extensible c-treeACE database technology this is no longer a challenge. FairCom DB API offers a flexible and powerful relational architecture over your existing c-treeACE data files. A feature within FairCom DB API called Virtual Tables allows an easy path to full relational support, including SQL, over these types of existing tables—all without changes to the underlying application and data.

Virtual Tables

A Virtual Table is operated upon as an ordinary FairCom DB API table (with a few limitations, depending on the type of the Virtual Table, see *Unsupported Functions* for details). However, it is not an actual physical table; it is an "interpretation" of the records (some or all) within the host (parent) table. Virtual Tables can be thought of as a generic concept. Actual implementations of this concept are the Virtual Table types, which are currently implemented through the *MultiRecordTable* type.

A Virtual Table is identified in the FairCom DB API database dictionary as a particular marker indicating the type of Virtual Table. The dictionary handling function has been modified to differentiate Virtual Tables from regular tables and also make the FairCom DB API interface view a Virtual Table as a regular table. For instance, when listing tables of a dictionary, Virtual Tables are listed together along with regular tables. Once created, a Virtual Table is operated upon as any other FairCom DB API table.

Multiple Record Table

MultiRecordTable, *MRTTable*, support is the first implementation of a Virtual Table. This is a type of table in which the record structure varies from record to record depending on some criteria. This implementation requires that within a given record schema there is a rule identifying if a record can be properly represented with that record schema or not. This support requires common schema to be defined for all records (however, this may describe only the first part of the record) and a filter identifying records belonging to a particular record schema based on this common schema.

The actual table which contains all data records is required to have a standard c-treeACE *DODA* and *IFIL* resource (or provided through external callbacks). This table is called the host or parent. The FairCom DB API API provides the means to add definitions for the various record type schemas by defining an *MRTTable* per each record type as a Virtual Table for the host table.

The FairCom DB API **ctdbCreateMRTTTable()** function behaves much like the ordinary **ctdbCreateTable()** function, however, it creates a *MRTTable* definition.

ctdbIsVTable(CTHANDLE table) can be used to check if a table is a Virtual Table or a regular table.

Virtual Table Callbacks

- CTDB_ON_TABLE_GET_VTABLE_INFO
  CTDB_ON_TABLE_GET_VTABLE_INFO is called when FairCom DB API looks for the Virtual Table resource, typically at the time of a Virtual Table open or when calling **ctdbGetVTableInfoFromTable()**. This callback should be implemented when using alternative (to the resource as defined) methods of tracking Virtual Table definitions. (For example, c-treeACE COBOL support implements this callback.)

API for Virtual Tables

The Function Reference appendix describes the FairCom DB API functions available to enable Virtual Table support:

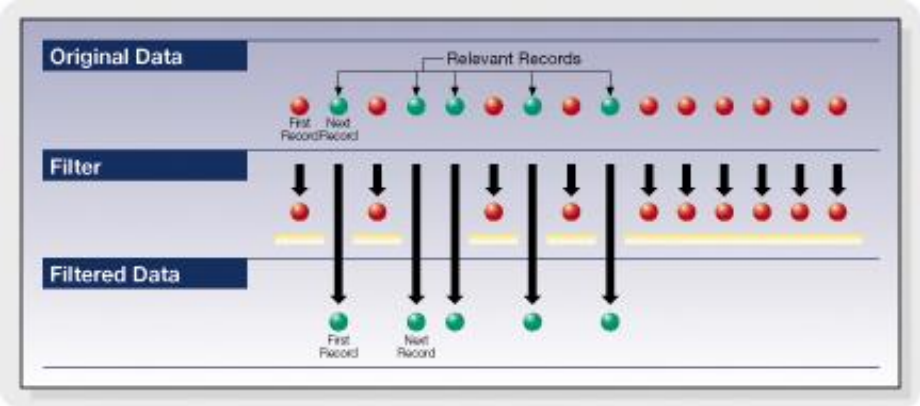
- **ctdbAddMRTTable()** (page 155)
- **ctdbAddVTableResource()** (page 157)
- **ctdbAllocVTableInfo()** (page 158)
- **ctdbCreateMRTTable()** (page 159)
- **ctdbFreeVTableInfo()** (page 161)
- **ctdbGetVTableInfoFromTable()** (page 162)
- **ctdbGetVTableNumber()** (page 163)
- **ctdbIsVTable()** (page 164)
- **ctdbRemoveVTableResource()** (page 165)
- **ctdbSetMRTTableFilter()** (page 166)

4.4 Data Filters

The data filter capability implemented with the **SetDataFilter()** function allows developers to access a qualified set of records specified by a conditional statement. Unlike the conditional index expressions, which become a permanent part of the index definition (except for temporary indexes), a specific filter is a “run-time” feature and remains active until you change filter definitions or the data file is closed. Further, a specific filter only applies to the calling user: it does not affect the retrieval of records by other users.

Filtering lets you specify criteria to temporarily restrict access only to records that meet the filter criteria. c-treeACE filters in client/server environment may substantially reduce network traffic since only the records that satisfy the filter condition will be returned. ISAM-level record requests, including sets and batches, skip over records failing the filter transparently to the user. In client/server, this may substantially reduce network traffic since only the records that satisfy the filter will be returned. For example, with a data file of 1,000 records, if a filter excludes 900 of the records, a **FirstRecord()** - **NextRecord()** loop results in only 100 calls to the c-treeACE Server, not 1,000 calls. Generally, an index can also be used to help reduce unnecessary record retrievals, however, the filter allows for as-needed retrievals.

With the **SetDataFilter()** function, record returns, such as from **FirstRecord()** or **GetRecord()**, can be limited to a specific range. The conditional expressions, in conjunction with the schema map and symbolic field names, describe the range of values returned. For example, the expression “(ZipCode >= 65000) && (ZipCode < 66000)” would limit returns to records with 65xxx ZipCodes. While there is a limit of one expression, the expressions can be as complex as necessary to accomplish your goals.



For partitioned files, call **SetDataFilter()** for the host data file and the filter automatically applies to each partition member file. See **SetDataFilter()** (https://docs.faircom.com/doc**/ctreeplus/setdatafilter.htm**) for additional details.

Data Filter Stacks - Support Multiple Data Record Filters per File

This is a significant enhancement to our existing data record filter support. Now you have the ability to “stack” data record filters over the same data file. The power this gives you is extensive.

Many times in an application’s logic, data-read requirements might be located in different areas of your code and necessitate different criteria. You might establish one data filter in one location and then later, in another set of logic, find the need for more filtering. Say at logon time, you establish a filter based on user, then somewhere later in the application, a date is introduced.

Now with the ability to “stack” your data file filters, you can add the required filter when you need it most. A new function, **SetDataFilterNbr()** has been created to use this feature.

Conditional Expression Support Enhanced

Reference Any Section of the Record

A new feature allows you to reference any section of the record (offset,length) even if it does not correspond to a field.

It can be very useful to be able to write a filter (or a conditional expression in general) that directly references a section of the record even if this section does not correspond to a field. For conditional expressions we now allow retrieving a portion of a record as it would be a CT_ARRAY field. The new function syntax is **FIELD(***offset,size***)** and evaluates as a CT_ARRAY field with length *size* containing the portion of the current record starting at offset *offset*.

Create a Filter on a Table without the DODA (or SCHEMA)

The set filter function, **SETFLTR()**, makes specific checks to ensure that “schema” and “names” are available (although they may be empty) before parsing a filter expression.

The filter expression parser (as well as the run-time logic) do not require the existence of schema and names as long as the filter expression does not reference fields. The check prevented you from running filters that do not reference field. We have relaxed this constraint in a way that if the “schema” is present then also “names” must be successfully retrieved otherwise it fails.

If no schema is present, we go directly to the filter parsing, allowing users to utilize filters on files with no schema, perhaps using our filter callback support.

Runtime Support Improved to Support IN and BETWEEN Operators

The IN and BETWEEN operators implemented within our conditional expression logic require additional support from FairCom engineers to utilize this power. If you are interested in this capability, please contact FairCom.

Filter Callback Support Enhanced

The ability to author a data record filter callback function is an extremely valuable feature for developers with custom filter needs. The following enhancements have been implemented to this support:

- Ability to enforce a callback DLL version so users do not utilize the wrong callback DLL.

- The c-treeACE Server DLL now exports the function **ctStatusLogWrite()**, which writes the specified message to the c-treeACE Server status log file.
- The name of the data file for which a filter callback function is set is now available to the DLL load, filter evaluation, and DLL unload functions exported by a data record filter callback DLL.

Data Record Filter Callback DLL Enhancements

The following modifications were made to the data record filter callback DLL support:

- Require a filter callback DLL to export a function named **GetFilterVersion()** that returns the version of the filter callback API that the DLL uses. When loading a filter callback DLL, the c-treeACE Server calls this function and fails to load the DLL with error **CBKV_ERR** (870) if the filter callback API version returned by the DLL doesn't match the filter callback DLL API version that the c-treeACE Server is using.
- Make the name of the data file for which a filter callback function is set available to the DLL load, filter evaluation, and DLL unload functions exported by a data record filter callback DLL. Note that this revision sets the c-treeACE Server's filter callback API version to 2 to indicate the addition of a filename field, *cbfilnam*, to the *CBDLL* structure. The name of the data file is copied to this field so that it is available to the data record filter DLL callback functions.
- Callback functions exported from a filter callback DLL were renamed:
  - **ctfiltercbLoadLib()** is now **LoadFilter()**
  - **ctfiltercbEval()** is now **EvaluateFilter()**
  - **ctfiltercbUnloadLib()** is now **UnloadFilter()**

  The *cblibhandle* field of the *CBDLL* structure was renamed *cbfilterhandle*.
- The c-treeACE Server DLL now exports the function **ctStatusLogWrite()**, which writes the specified message to the c-treeACE Server status log file. The prototype for this function is:

  ```
  COUNT ctStatusLogWrite(pTEXT msg,NINT err)
  ```

  - *msg* is the message to write to the status log
  - *err* is the error code to include in the message.

  Example

  ```
  ctStatusLogWrite("This is a test", 3);
  ```

  writes a message in the following format to *CTSTATUS.FCS*:

  ```
  Wed Jun 10 15:10:27 2009
   - User# 00011    This is a test: 3
  ```

4.5 File Open Performance

We have introduced a new capability that allows files to remain open. Our new *KEEPOPEN* file support avoids losing the data/index cache entries associated with the file. Subsequent users can open the file quicker and operate on the file with the benefits of an existing cache. This can result in significantly better performance when working with large numbers of files, especially in "stateless" systems such as SQL and web applications. The c-treeACE Server automatically closes *KEEPOPEN* files that are not in use when a file create or open operation finds that no more file control blocks are available. And when the number of cache pages in use by a *KEEPOPEN* file that is not in use drops to zero, an administrative thread closes the file.

KEEPOPEN - Enhanced File Open Performance

Memory files were designed with the option of remaining open after all users have closed the file. This support was required to avoid losing the contents of the file such that subsequent users can open the file and read and/or update the contents. This mode is referred to as *KEEPOPEN*.

When a non-memory file is normally physically closed, c-treeACE removes the data cache and/or index buffer entries associated with the file. A file is physically closed when all users that have the file open close the file.

Allow Files to Remain Open Even When Not Used

This *KEEPOPEN* support has been extended to all physical ISAM data files and their associated indexes. An advantage of this mode is to keep the files in the data cache and index buffers, even after all users have closed the file. Subsequent file opens benefit from having the cache contents immediately available. It also eliminates a physical open when the next user opens the file. This can result in significantly better performance when working with large numbers of files.

Files to be treated in this manner are specified in the server configuration file with one or more entries of the form:

```
KEEPOPEN_LIST  <file spec>
```

where file *<file spec>* can be a file name or a partial name including wild card characters.

Upon file creation or physical open, the *KEEPOPEN* mode is applied to the data file and its indexes if:

1. the file name matches a *<file spec>*, and;
2. the file is a data file, and;
3. the data file creation or open is part of an ISAM creation or open.

If all users have closed a *KEEPOPEN* file, the **ctCLSNAM()** API function can be called to close the data file and its associated indexes.

Additional Notes

- Superfile ISAM members are also supported with this mode. A Superfile host can be created/opened with the *KEEPOPEN* attribute if it is created/opened as an ISAM file (without indexes).
- If indexes are added to an ISAM data file with the *KEEPOPEN* attribute, then the new indexes inherit the *KEEPOPEN* attribute. Regardless of *KEEPOPEN* attributes, dropped indexes are either deleted or renamed depending on their *TRANDEP* attributes. If the index is renamed (*RSTRDEL*) then even if the index had *KEEPOPEN* attributes, the index is physically closed.

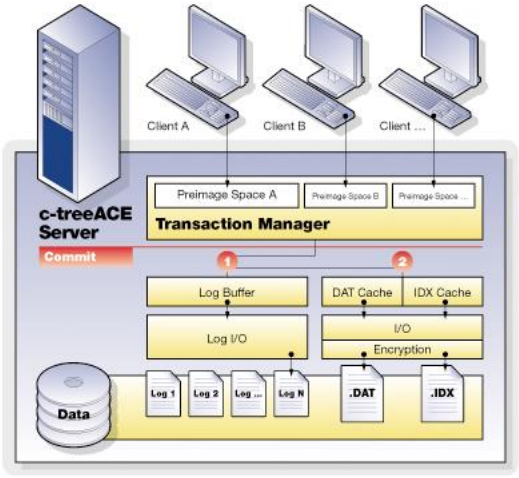
Extended KEEPOPEN File Mode for Fast File Access

One side effect of keeping files open is that a c-treeACE file control block is consumed. If the file is not explicitly closed, these resources are not freed. c-treeACE now automatically closes *KEEPOPEN* files that are not in use when a file create or open operation finds that no more file control blocks are available. Further, when the number of cache pages in use by a *KEEPOPEN* file that is not in use drops to zero, an administrative thread closes the file.

The create and open functions now check if file control blocks are consumed. If so, an attempt is made to determine if a *KEEPOPEN* file is available to close. If so, the file is closed, and the operation is retried. The c-treeACE configuration option `KEEPOPEN_CLOSE_RETRY_LIMIT` determines the number of times these functions retry the operation before failing. This value defaults to 3.

4.6 Transaction Processing

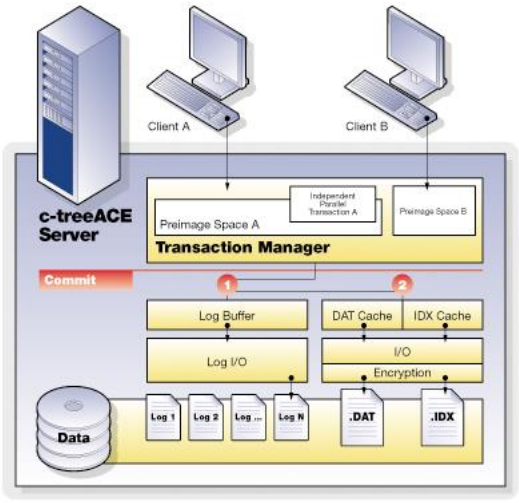
c-treeACE is a highly tunable database engine that offers industrial quality on-line transaction processing (OLTP) features. These features guarantee ACID (atomicity, consistency, isolation, durability) properties of transactions. Multiple save/restore points are also supported ensuring maximum flexibility and integrity of your data. Rollback and restore to any portion of a transaction without abandoning the entire transaction.



c-treeACE transaction processing is implemented on three levels: Full, PreImage and No Transaction support. Each level offers different features and benefits to enhance overall database performance.

Independent Parallel Transactions

This is a perfect example of FairCom responding to a request from a customer. We have added the ability to perform a type of Independent Parallel Transaction. In traditional database methodology, this ability might not exist, yet special customer circumstance justified the implementation of this new feature. After you have “begun” a transaction (Transaction A), and then performed some operations, you may now “begin” what we call an Independent Parallel Transaction (Transaction B). When this secondary transaction is committed or aborted, only the operations performed within that transaction are affected. Operations performed within the primary transaction (Transaction A) retain their traditional scope.



In one customer’s case, they wanted to create new files while inside a transaction. These new files were not necessarily part of the atomicity requirements of the primary transaction, yet if the primary transaction was aborted, the creation of these file was also being reverted. Now, within the primary transaction, the customer initiates an Independent Parallel Transaction, performs the create for the files, and then commits the Independent Parallel Transaction. If the primary transaction is later aborted, the abort does not affect these newly created files.

IICT

The Immediate Independent Commit Transaction, *IICT*, permits a thread with an active, pending transaction to also execute immediate commit transactions, even on the same physical file that

may have been updated by the still pending (regular) transaction. An *IICT* is essentially an auto commit ISAM update, but with the added characteristic that an *IICT* can be executed even while a transaction is pending for the same user (thread).

It is important to note that the *IICT* is independent of the existing transaction: it is as if another user/thread is executing the *IICT*. The following pseudo code example demonstrates this independence:

Example

1. Begin transaction
2. ISAM add record R1 with unique key U to file F
3. Switch to *IICT* mode
4. ISAM add record R2 with unique key U to file F: returns error **TPND_ERR** (420)

If we did not switch to *IICT* mode, the second add would have failed with a **KDUP_ERR** (2); however, the *IICT* mode made the second add use a separate transaction and the second add found a pending add for key U, hence the **TPND_ERR**. Just as if another thread had a key U add pending.

A data file and it's associated indices are put into *IICT* mode with a call

```
PUTHDR(datno,1,ctIICThdr)
```

and are restored to regular mode with a call

```
PUTHDR(datno,0,ctIICThdr)
```

It is possible in c-tree for a thread to open the same file in shared mode more than once, each open using a different user file number. And it is possible to put one or more of these files in *IICT* mode while the remaining files stay in regular mode.

Note: If a file has been opened more than once by the same thread, then the updates within a (regular) transaction made to the different file numbers are treated the same as if only one open had occurred.

These special *filno* values enable specific *IICT* operations:

- *ctIICTbegin* -1
- *ctIICTcommit* -2
- *ctIICTabort* -3

Auto Recovery

Recovery in Alternate Locations with REDIRECT

The REDIRECT feature is a useful feature allowing a file originating in one directory structure to be repositioned into another directory location during dynamic dump restore. This support has been extended to c-treeACE automatic recovery.

Redirection rules can be specified by using the following configuration entry one or more times in the server configuration file *ctsrvr.cfg*:

```
REDIRECT <old path> <new path>
```

The `REDIRECT` entry redirects filename references in the transaction logs during automatic recovery to the specified new filename. This option is useful when c-treeACE data and index files are moved to a different location (on the same system or on another system) before running automatic recovery.

To specify an empty string for one of the `REDIRECT` arguments use a pair of double quotes ("").

Examples

If a file originally existed with the name and path *C:\Documents and Settings\Administrator\c-tree Data\customer.dat* and now exists as the file *D:\Documents and Settings\Guest\customer.dat*, the following option will allow automatic recovery to proceed and find the file in its new location:

```
REDIRECT "C:\Documents and Settings\Administrator\c-tree Data" "D:\Documents and Settings\Guest"
```

Here's a similar example using Unix paths, where the original file is named */users/administrator/c-tree data/customer.dat* and the file now exists as */users/guest/customer.dat*:

```
REDIRECT "/users/administrator/c-tree data" "/users/guest"
```

Note: Use double quotes when a filename contains spaces.

Updating IFIL Filenames

As a result of redirection, if the *IFIL* resource of the file contained a path, this path would be incorrect after the file was redirected to the new location. To support copying c-treeACE files from one directory location to another (on the same system or on a different system) and accessing them in their new location, it is necessary to update any filename paths in a c-treeACE data file's IFIL resource.

The c-treeACE configuration option `REDIRECT_IFIL <filename>` provides support for automatically modifying redirected files on the server. When this option is specified, on server start up (after automatic recovery completes) the file named <filename> is opened and its list of file names is read from it. <filename> is a text file containing one c-treeACE data file per line. For each file specified in <filename> c-treeACE opens the file and uses the filename redirection rules (specified with one or more of the `REDIRECT` options) to change the data and index file paths in the IFIL resource of the file.

Refer to the c-treeACE **ctredirect** standalone utility to manually modify files that may have been moved.

Improved Automatic Recovery Performance

A number of improvements to the performance of Automatic Recovery are included in this release:

- **TRANSACTION_FLUSH** - In normal operation, the TRANSACTION_FLUSH keyword controls for the maximum number of updates to a data or index buffer before it is flushed. It was found that during automatic recovery, a small (default) TRANSACTION_FLUSH setting would cause recovery to proceed much slower on very large data sets. Investigation into the automatic recovery operations revealed key insert and delete operations (such as when loading key values into an index or when redoing or undoing transactions) use the specified

transaction flush value when determining to update an index buffer. These buffers could then be frequently written to disk, causing a noticeable performance impact. Increasing the flush value reduced this frequency. Modifications were made to completely skip the buffer write to disk during recovery operations as all files are closed at the end of recovery and the intermediate writes are unnecessary. The result is a much improved recovery when a large amount of data is retained in the transaction logs.

- **Skip duplicate SetNodeName entries** - c-treeACE now skips duplicate SetNodeName transaction log entries for faster performance. While analyzing transaction logs for automatic recovery performance, multiple duplicate entries for setting a client node name were found. We now avoid the extraneous “set-node-name” entries in the transaction logs.
- **RECOVER_MEMLOG for faster automatic recovery with Advanced Encryption** - When transaction logs are encrypted with advanced encryption enabled, FairCom recommends the use of the RECOVER_MEMLOG configuration option to load the transaction logs into memory before recovery. This provides a faster decryption as data is already decrypted upon loading of the transaction log.
- **Automatic recovery for Partitioned Files** - Significant improvements overall for Partition File support are reflected in this release. This includes both the performance and operations on partition files within auto recovery. It also includes support for partition file’s new global unique indexes.
- **Recognize inaccessible files** - Automatic recovery can now recognize that a c-treeACE data or index file that is required by automatic recovery exists but is not accessible. In this situation, automatic recovery logs an error message listing the name of the inaccessible files and terminates. If it is desired for some reason to skip files that are not accessible, the keyword SKIP_INACCESSIBLE_FILES YES causes automatic recovery to skip any file that is not accessible and continue forward with the automatic recovery.
- **Detect infinite loop during recovery** - We resolved an issue when recovery appears to hang with 100 percent CPU usage. If a loop exists in the delete stack, we now detect it resolving an infinite loop situation. Based on the file size on disk, set an upper limit on the number of index nodes that could be in the delete stack; we truncate the delete stack if this limit is exceeded.

Additional Recovery Logging

Automatic Recovery is used to bring transaction controlled c-treeACE data and index files back to a consistent state in the event of an unplanned server outage. The recovery process occurs over several phases. Various details of these phases can be observed with the `RECOVER_DETAILS YES` server configuration option. This logging of details is now on by default, and `RECOVERY_DETAILS NO` can be used to suppress this logging.

c-treeACE recovery has been further modified to enhance the logging of details with additional messages specifying phase and progress during the automatic recovery process. Index composition can frequently be a phase that takes the longest amount of recovery time. As it is useful to know the proceedings of this phase, specific messages are now output describing the progress of this phase, and the index member involved. These descriptions are output whether or not the *LOGIDX* option is in force.

Below is an example of messages that can be found in *CTSTATUS.FCS* when *LOGIDX* is not used during automatic recovery. The description in square brackets indicates why *LOGIDX* was not used:

```
Mon Nov 23 09:32:44 2009
 - User# 00001      Index repair time:   0 seconds.
Mon Nov 23 09:32:49 2009
 - User# 00001      tranrcv: Reconstructing index mark.idx [LOGIDX not in file header]
Mon Nov 23 09:32:51 2009
 - User# 00001      tranrcv: Reconstructing index mark.idx M#01 [LOGIDX not in file header]
Mon Nov 23 09:32:52 2009
 - User# 00001      tranrcv: Reconstructing index mark.idx M#02 [LOGIDX not in file header]
Mon Nov 23 09:32:53 2009
 - User# 00001      Index composition time:   9 seconds.
```

Below is an example of messages found in *CTSTATUS.FCS* when *LOGIDX* is used during automatic recovery:

```
Mon Nov 23 10:46:26 2009
 - User# 00001  Index repair time:         0 second(s) for   1 repair(s).
Mon Nov 23 10:46:26 2009
 - User# 00001  tranrcv: Recomposing index file FAIRCOM.FCS DI:
Mon Nov 23 10:46:26 2009
 - User# 00001  tranrcv:     Processing abort node list entries.
Mon Nov 23 10:46:26 2009
 - User# 00001  tranrcv: Recomposing index file mark.idx:
Mon Nov 23 10:46:26 2009
 - User# 00001  tranrcv:     Processing LOGIDX node entries.
Mon Nov 23 10:46:26 2009
 - User# 00001  tranrcv:     Checking index delete stack.
Mon Nov 23 10:46:26 2009
 - User# 00001  tranrcv: Recomposing index file mark.idx M#01:
Mon Nov 23 10:46:26 2009
 - User# 00001  tranrcv:     Processing LOGIDX node entries.
Mon Nov 23 10:46:26 2009
 - User# 00001  tranrcv: Recomposing index file mark.idx M#02:
Mon Nov 23 10:46:26 2009
 - User# 00001  tranrcv:     Processing LOGIDX node entries.
Mon Nov 23 10:46:26 2009
 - User# 00001  Index composition time:    0 second(s).
```

Persistent Lock Behavior inside Transactions

Inside of a transaction, an unlock request on a record behaves differently depending on whether or not the record has been updated. If it has not been updated the unlock request succeeds. If it has been updated, the unlock returns a **NO_ERROR**, but the lock remains and the sysiocod is set to UDLK_TRN(-3). It is now possible to specify that locks persist even if the record has not been updated: The unlock will return **NO_ERROR** and sysiocod will be set to UDLK_TRN.

A user can turn on this state for all files used by the user with a call to **ctLOKDYN(***ctLOKDYNtranPersist***)**. Or the user can make a call **PUTHDR(***datno,1,ctTRNPERShdr***)** for particular data files for which the new state will be activated. If *ctLOKDYNtranPersist* is not enabled, then the individual file states prevail. However, if *ctLOKDYNtranPersist* is enabled, the individual file states are ignored.

A call to **ctLOKDYN(***ctLOKDYNnopersistTran***)** will disable the user-wide state. A call to **PUTHDR(***datno,0,ctTRNPERShdr***)** will turn off the file state. The “tranPersist” state (whether

user-wide or file specific) only affects behavior at the time an unlock request is made. Whether it is on or off at the time the lock is granted does not affect unlock behavior.

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

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

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

Configurable Transaction Number Overflow Warning Limit

When c-treeACE supports 6-byte transaction numbers it does not display transaction overflow warnings until the current transaction number approaches the 6-byte transaction number limit. But if 4-byte transaction number files are in use, a key insert or delete will fail if the current transaction number exceeds the 4-byte transaction number limit (however, c-treeACE will continue operating).

To allow a server administrator to determine when the server’s transaction number is approaching the 4-byte transaction number limit, the following configuration option was added:

```
TRAN_OVERFLOW_THRESHOLD <transaction_number>
```

This keyword causes the c-tree Server to log the following warning message to *CTSTATUS.FCS* and to standard output (or the message monitor window on Windows systems) when the current transaction number exceeds the specified transaction number:

```
WARNING: The current transaction number (####) exceeds the user-defined threshold.
```

The message is logged every 10000 transactions once this limit has been reached. The `TRAN_OVERFLOW_THRESHOLD` limit can be set to any value up to 0x3fffffffffff, which is the highest 6-byte transaction number that c-treeACE supports.

Other Transaction Issues

Additional details related to transaction processing have been addressed in this release, including:

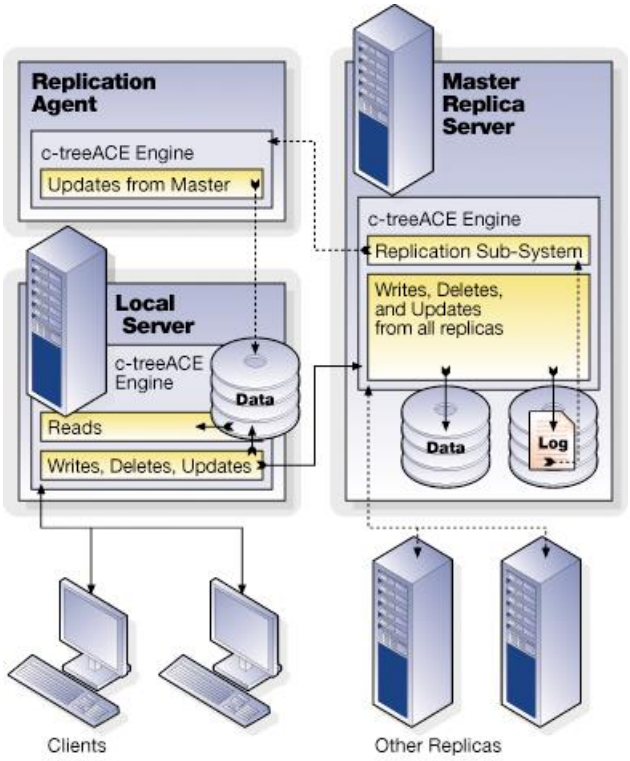
- **Global Transaction Number** - The c-treeACE Server utilizes a “Global Transaction Number” value to determine the next transaction number when a new transaction is created. Transaction numbers are critical to the data integrity, and, among other areas, are persisted within our transaction logs. Legacy versions of the c-treeACE Server (V8.14 and prior) maintained a 4-byte global transaction number. Our current server maintains a 6-byte transaction number. If the number of transactions initiated for a database exceeds this 6-byte

number, transaction log maintenance must be performed. This involves shutting down the server and executing a maintenance utility on the database files and logs. With a 6-byte number, a server can perform approximately 1000 transactions per second for 2000 years. That's 70 TRILLION transactions, so typically this offline maintenance is never required. We have added the ability for the users to define within the server's configuration file a "threshold warning" value. When the Global Transaction Number base reaches this value, the server will issue warnings. This user-defined value allows users who still have legacy data with 4-byte transactions numbers to set the "warning threshold" below the 4-byte limit, which is around 1 billion transactions, giving notification of the necessary maintenance.

- **Transactions with over 32768 SAVE POINTS** - It may seem a little extensive to have this many save-points within a single transaction, yet c-treeACE developers are known for pushing the limits.
- **Overlapping checkpoints** - During our performance profiling, we improved the synchronization of a user-invoked checkpoint with an internal checkpoint already in operations. If a checkpoint was forced from an application level **CTCHKPNT()** call, and an in-process checkpoint is already in progress, internal logic has been improved to avoid any additional processing overhead as a checkpoint is already in progress.
- **Deferred transaction file number assignment** - An additional performance improvement has been implemented. We found that transaction file numbers could be deferred until the file was actually updated. Previously, when c-treeACE physically opened a transaction-controlled data or index file, it assigned a transaction file number to the file and wrote an entry to the transaction log. As a result, even if a file was simply opened, read, and then closed, the operations performed unnecessary log overhead. Now, c-treeACE defers the file number assignment and transaction logging until the first update is made to the file. This gives us performance gains for these "read-only" situations.
- **Excessive number of transaction logs** - Under rare situations, the c-treeACE Server had been observed to retain an excessive number of transaction logs. While numerous logs are expected in normal processing under a large transaction loads, we have also determined that a situation may exist when a buffer page has not been flushed, causing the server to retain a hold on a log file when it does not need to do so. The c-treeACE Server now detects and corrects this situation during checkpoint processing.

4.7 Replication

Replication is an important technology. c-treeACE's current replication support is built-in at the "low-level," so it is designed for high-speed, and may be used to support high availability, redundant resources, improve reliability, and fault-tolerance requirements. We offer a strong low-level infrastructure and provide the necessary engineering expertise to advise our customers with the implementation of this technology.



Significant improvements are reflected in this release, including:

- **Mapping file names** - Replication Agent supports mapping the names of c-treeACE data files on a source server to different filenames on a target server.
- **Windows service** - Support for running Replication Agent as a Windows service and writing to the Windows event log.
- **Pause and resume** - Added pause/resume options to the Replication Agent.
- **Persisted replication states** - Allows the Replication Agent to always know the last transaction committed to the target.
- **Compressed data** - Support replication of compressed data.
- **Partitioned files** - Support replicating data in partitioned files.
- **Failed file opens** - Improved Replication Agent exception handling with failed file opens.
- **Record locking** - Support configuration options to specify how many times to attempt to lock a record and how much time to sleep between record lock attempts.
- **Auto-numbering** - Support for replicating auto-numbered records.

Replication State Persistence

If the Replication Agent process terminates abnormally, it is desirable for the agent to resume replicating transactions from where it left off in the source server transaction logs. A persistence state is needed to maintain this information. This persistence requires that the Replication Agent always knows the last transaction it has committed to the target FairCom Server.

To enable this persistent state, a special table is stored on the target server for this purpose, and for each transaction the Replication Agent commits it updates a replication state record on the target server in that same transaction. This way, the Replication Agent can always read its latest

state from the target server, even if the Replication Agent process terminates, or should it lose its connection to the target server, or the target server terminates and restarts.

When multiple Replication Agents are connected to a single target server, it is necessary to maintain state information unique to each replication instance. To do this, a configurable unique ID can be assigned to each agent. The Replication Agent unique ID is a string (maximum 32 characters). The Replication Agent's configuration file, *ctreplagent.cfg*, can specify the unique ID using the `unique_id` configuration option. For example:

```
unique_id myreplagent
```

If not specified, the unique ID defaults to REPLAGENT.

Target Server Files

Once the replication has started and successfully connected to both the source and target servers, it will create a set of files on the target server which contains information about the current state and position of replication within the source transaction logs. This allows the Replication Agent to pick up from a previous session should a network connection fail, or the agent is paused for administrative purposes. These files are:

REPLSTATEDT.FCS
REPLSTATEIX.FCS

REPLSTATEDT.FCS and is a variable-length data file that contains one record for each Replication Agent that has registered itself with the target server. The records are indexed by the Replication Agent unique ID.

When the Replication Agent starts, it reads its unique ID from the *ctreplagent.cfg* configuration file, connects to the target server, creates *REPLSTATEDT.FCS* if it does not exist, and then reads its Replication Agent state record.

The Replication Agent opens the file *ctreplagent.ini* if it exists and reads the saved replication state (log read position and last commit log position). The state from *ctreplagent.ini* is used if that file exists; otherwise the state from the Replication Agent state record is used. If neither exists, the Replication Agent commences its scan from log 1 position 0.

Note: If log 1 no longer exists (for example, if the transaction files were not removed along with the removal of the *REPLAGENT.FCS* file), then error 96 (**LOPN_ERR**, Log file/Start file open error) is possible. In this case, existing transaction logs will need to be removed, or a *ctreplagent.ini* file needs to be created with correct log and position information.

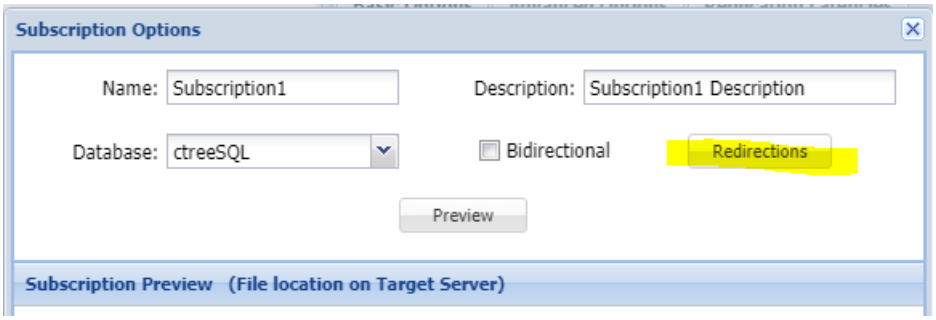
When committing a transaction on the target server, the Replication Agent writes its current state to the replication state record before it commits the transaction. Note that the Replication Agent reads its replication state record when it connects to the target server, and the Replication Agent keeps that record locked until it disconnects from the target server.

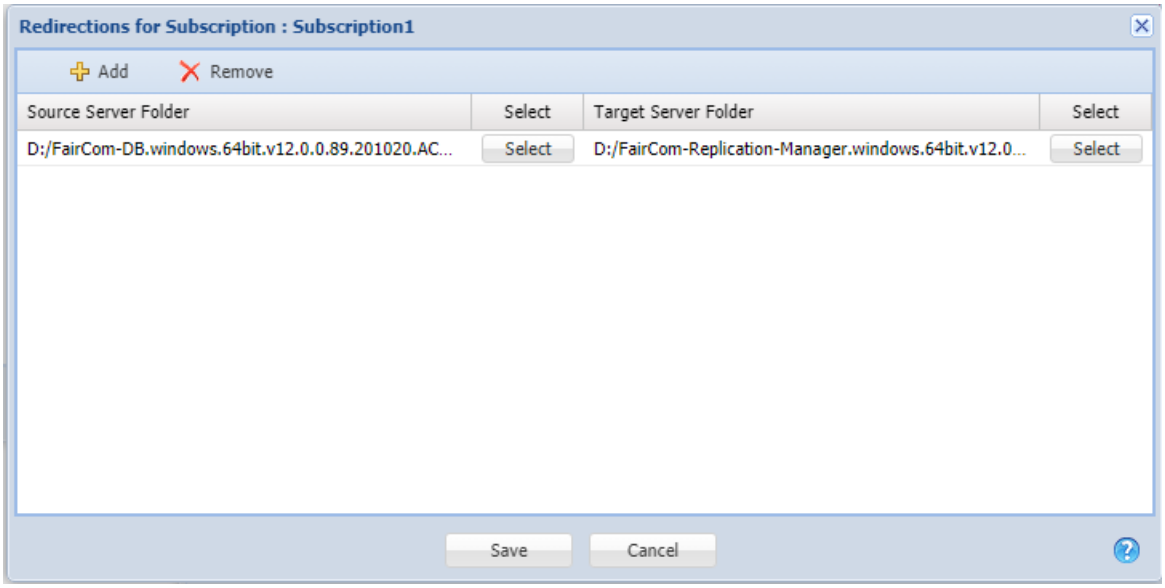
For more, see **Starting Replication from a Known Log Position** in the replication documentation.

REDIRECT Option for Replication Agent to Alternate Destinations

The Replication Agent supports mapping the names of c-tree data files on a source server to different filenames on a target server.

From Replication Manager select your path configurations from the Subscription options.





Manual Configuration

The configuration option `redirect <old> <new>` can be specified one or more times in the Replication Agent configuration file, *ctreplagent.cfg*. *<old>* specifies the name of a replicated data file on the c-treeACE source server, and *<new>* specifies the name of the corresponding data file on the c-treeACE target server. This option forces the Replication Agent to replace the old name with the new name when opening the file on the target server.

Examples

Consider a file that exists on the FairCom source with the name *C:\Documents and Settings\Administrator\c-tree Data\customer.dat* and exists on the FairCom target as the file *D:\Documents and Settings\Guest\customer.dat*, the following option allows the Replication Agent to open the file in its location on the target system:

```
redirect "C:\Documents and Settings\Administrator\c-tree Data\customer.dat" "D:\Documents and
Settings\Guest\customer.dat"
```

This option can specify a portion of the filename. For example, to redirect the names of all files in a particular directory on the source system to another directory on the target system you could use:

```
redirect olddir newdir
REDIRECT "C:\Documents and Settings\Administrator\c-tree Data" "D:\Documents and Settings\Guest"
```

Note: Use double quotes when a filename contains spaces.

Record Lock Error Retry and Diagnostics

FairCom Replication supports configuration options to specify how many times to attempt to lock a record and how much time to sleep between record lock attempts. These options are used when the FairCom Replication attempts to update a record on the target FairCom Server.

The option `lock_retry_count <count>` specified in *ctreplagent.cfg* indicates that a record read or update that fails with error **DLOK_ERR** (42, Could not obtain data record lock) is retried up to *<count>* times (default 2). The option `lock_retry_sleep <sleep_ms>` specified in *ctreplagent.cfg* indicates that before retrying the operation that failed with error **DLOK_ERR**, the Replication Agent sleeps for *<sleep_ms>* milliseconds (default 100).

When an update fails with **DLOK_ERR** (after exhausting the retries), the Replication Agent then logs the following message to *ctreplagent.log* as this error is not expected due to FairCom Replication usage of blocking locks:

```
ERR: Unexpectedly failed to update record: error code=42 (diag=<diagnostic_code>)
```

where *<diagnostic_code>* is one of the following:

1. **EQLVREC()** call failed with error **DLOK_ERR**
2. **RWTVREC()** call failed with error **DLOK_ERR**
3. **EQLREC()** call failed with error **DLOK_ERR**
4. **RWTREC()** call failed with error **DLOK_ERR**

c-treeACE was also modified to add diagnostic log messages in the function that is used to extend a file. That function contains logic that attempts to acquire a lock on new space. The function tries to acquire a lock up to 100 times, sleeping for 10 milliseconds between each lock attempt. It is hypothesized that a **RWTVREC()** operation could be failing with the **DLOK_ERR** error as this code exhausts its retry attempts. To determine if this is the case, additional logging was added with the following message to *CTSTATUS.FCS* when the lock attempt in this function fails:

```
extfil: Failed to lock offset 0x<offset> when extending file <filename>: 42
```

A record lock error (e.g., error **42**), may indicate the record on the target is already locked by another client. Check to see if a "rogue" replication accidentally running in background. You can use FairCom Monitor to view current connections and their origins. The command-line **ctadmn** utility can also provide this information.

Renamed Replication Agent Configuration Options

The term “master” has been renamed to “source” and “local” to “target” where it appears in the Replication Agent source code. As a result the following options have been renamed. Note that the original names may still be used for backward compatibility.

- `local_authfile` is now `target_authfile`
- `local_server` is now `target_server`
- `master_authfile` is now `source_authfile`
- `master_server` is now `source_server`

4.8 Backup/Restore

Volume Shadow Copy Service (VSS) Integration

The Volume Shadow Copy Service (VSS) writer has been added as an integral component of the c-treeACE Server for Windows. This component is supplied as a Windows dynamic link library (*c-treeACEVSSWriter.dll*) and can be optionally loaded by the c-treeACE Server at startup.

The Volume Shadow Service is a Microsoft technology built into Microsoft Windows operating systems starting from Microsoft Windows XP and later that allows applications to access a "point-in-time" snapshot of a logical drive. This service allows taking manual or automatic backup copies or "snapshots" of data. Snapshots have two primary purposes: they allow the creation of consistent backups of a volume, ensuring that the contents cannot change while the backup is being made; and they avoid problems with file locking. By creating a read-only copy of the volume, backup programs are able to access every file without interfering with other programs writing to those same files.

c-treeACE Server provides support for VSS through its VSS writer, which controls how c-treeACE data is set to a consistent state at the beginning of a VSS operation and maintain that consistency throughout the process.

Windows Volume Shadow Copy Service (VSS) Writer

The VSS writer has been added as an integral component of the c-treeACE Server for Windows. This component is supplied as a Windows dynamic link library (*c-treeACEVSSWriter.dll*) and can be optionally loaded by c-treeACE at startup. (An executable, *c-treeACEVSSWriter.exe*, is also supplied for testing purposes, however, not intended for production use.)

VSS Configuration

```
VSS_WRITER YES
```

With this option enabled, c-treeACE loads the Volume Shadow Copy Service (VSS) writer DLL (*c-treeACEVSSWriter.dll*) and initializes the VSS writer when the server starts. A FULL CONSISTENCY VSS backup of FairCom database files will be created, which is comparable to cleanly shutting down the FairCom server and then backing up all files.

For application files that are under full transaction control, a FULL CONSISTENCY VSS backup is more time-consuming and invasive than strictly required. Application files with the ctTRNLOG property are under full transaction control. If full transaction controlled files and the database transaction logs (L*.FCS and S0*.FCS) are included in backups, a normal database automatic recovery takes place at startup and will properly restore CRASH CONSISTENT backed up data. Do NOT enable VSS_WRITER YES in ctsrvr.cfg, on application files that are under full transaction control.

Note: VSS backups require the Volume Shadow Copy service to be running. If this Windows service is set to start manually or is off by default, it needs to be started before VSS backup will work.

The following message is logged in *CTSTATUS.FCS* indicating the VSS writer has been started:

```
Mon Sep 13 14:11:27 2010
 - User# 00001      VSS Init: Successfully started the VSS writer.
```

If you run the command "`vssadmin list writers`" on a machine with c-treeACE Server running and a properly configured VSS, the list should include `c-treeACEVSSWriter`.

Compatibility Notes

- The FairCom VSS writer is compatible with the backup utilities provided in the *server* versions of Windows.
- The Windows backup software provided in *desktop* versions of Windows (the Enterprise edition of Windows 7 and Windows 8) is not a VSS-compatible backup provider and therefore will *not* work with the FairCom VSS writer.
- Windows Server backup (2008 & 2012) is a VSS provider and works with the FairCom VSS writer.
- Acronis Backup has been tested on Windows 7 (both 32-bit and 64-bit) and works correctly with the FairCom VSS writer when configured with *ctsrvr.dds*.
- The Novastor backup utility has been tested on non-server versions of Windows and works correctly with the FairCom VSS writer when configured with *ctsrvr.dds*.
- Other third-party backup utilities may work with the FairCom VSS writer if they are VSS-compatible backup providers. Please check with the manufacturer of your backup utility for information about VSS compatibility.

User Permissions

FairCom VSS Writer is intended to be run by users with Administrator permissions. To avoid permission issues when running the VSS Writer with a user that is not an Administrator, you must perform the following operations on the Windows registry:

1. Run the Windows **regedit** command.
2. Browse to find the Key:
   HKEY_LOCAL_MACHINE>SYSTEM>CurrentControlSet>Services>VSS>VssAccessControl
3. Insert a new REG_DWORD value with the following syntax DOMAINNAME\USERNAME. For example, if your domain is MYDomain and your user name is User, enter: `MYDomain\User`
4. Set the newly created key to the hexadecimal value of `1`.
5. Restart the computer to apply the changes.

Files to Be Backed Up

The VSS writer needs a list of files that are considered as under the server's control. This information must be located in the file *ctsrvr.dds* residing in the server's working directory (where the **faircom.exe** is located). For the VSS backup, only entries between `!FILES` and `!END` are relevant. There is no directory recursion, so wildcards will not be matched in subdirectories.

```
!FILES
C:\FairCom\ctreeSDK\ctreeAPI\bin.sql\ctreeSQL.dbs\test1.dat
C:\FairCom\ctreeSDK\ctreeAPI\bin.sql\ctreeSQL.dbs\test1.idx
ctreeSQL.dbs\*.dat
ctreeSQL.dbs\*.idx
ctreeSQL.dbs\SQL_SYS\*
!END
```

This information tells the backup utility which files are under c-treeACE control. If the set of files being backed up does not intersect with the set of files listed in *ctsrvr.dds*, the VSS service does not interact with c-treeACE VSS writer, resulting in an invalid backup of any files open by the server.

While testing, it is recommended to run the c-treeACE SQL Server with `DIAGNOSTICS VSS_WRITER` in *ctsrvr.cfg*. When the VSS writer is correctly configured, you should see entries logged to *CTSTATUS.FCS* like those listed in *VSS Diagnostic Logging*.

VSS Diagnostic Logging

The following c-treeACE configuration option enables VSS writer diagnostic logging (this can be enabled dynamically on the fly with server administrator utilities):

```
DIAGNOSTICS VSS_WRITER
```

When enabled, the VSS writer logs diagnostic messages to *CTSTATUS.FCS*. These messages indicate the sequence of operations to which the VSS writer is responding. Some examples are shown below:

```
Tue Sep 14 15:44:05 2010
 - User# 00016      VSS Diag: [0x1098]     c-treeACEVSSWriter::OnIdentify called
Tue Sep 14 15:44:07 2010
 - User# 00016      VSS Diag: [0x1098]     c-treeACEVSSWriter::OnPrepareBackup called
Tue Sep 14 15:44:07 2010
 - User# 00016      VSS Diag: [0x1098] (+) Component: CtreeACE
Tue Sep 14 15:44:07 2010
 - User# 00016      VSS Diag: [0x1098]     c-treeACEVSSWriter::OnPrepareSnapshot called
Tue Sep 14 15:44:07 2010
 - User# 00016      VSS Diag: [0x1098]     c-treeACEVSSWriter::OnFreeze called
Tue Sep 14 15:44:07 2010
 - User# 00016      VSS Diag: [0x1098]       QuietCtree(ctQTblockALL | ctQTflushAllFiles)...
Tue Sep 14 15:44:08 2010
 - User# 00016      VSS Diag: [0x1098]     c-treeACEVSSWriter::OnThaw called
Tue Sep 14 15:44:08 2010
 - User# 00016      VSS Diag: [0x1098]       QuietCtree(ctQTunblockALL)...
Tue Sep 14 15:44:08 2010
 - User# 00016      VSS Diag: [0x1098]     c-treeACEVSSWriter::OnPostSnapshot called
Tue Sep 14 15:44:10 2010
 - User# 00016      VSS Diag: [0x1098]     c-treeACEVSSWriter::OnIdentify called
Tue Sep 14 15:44:25 2010
 - User# 00016      VSS Diag: [0x1098]     c-treeACEVSSWriter::OnIdentify called
Tue Sep 14 15:44:26 2010
 - User# 00016      VSS Diag: [0x1098]     c-treeACEVSSWriter::OnBackupComplete called
Tue Sep 14 15:44:26 2010
 - User# 00016      VSS Diag: [0x1098]     c-treeACEVSSWriter::OnBackupShutdown called
```

Note: The VSS writer always logs error messages to the Windows event log, even if the `DIAGNOSTICS VSS_WRITER` option is not specified in the configuration file.

Dynamic API Interaction

The following SQL built-in procedures can enable VSS Writer support:

```
call  fc_set_sysconfig('vss_writer',  'YES');
call  fc_set_sysconfig('diagnostics', 'VSS_WRITER');
```

The **SetConfiguration()**, **ctSETCFG()** API function can be used to change VSS configuration dynamically.

Examples

If the c-treeACE VSS writer is not running, the following call starts it:

```
    ctSETCFG(setcfgVSS_WRITER, "YES");
```

If the c-treeACE VSS writer is running, the following call stops it:

```
    ctSETCFG(setcfgVSS_WRITER, "NO");
```

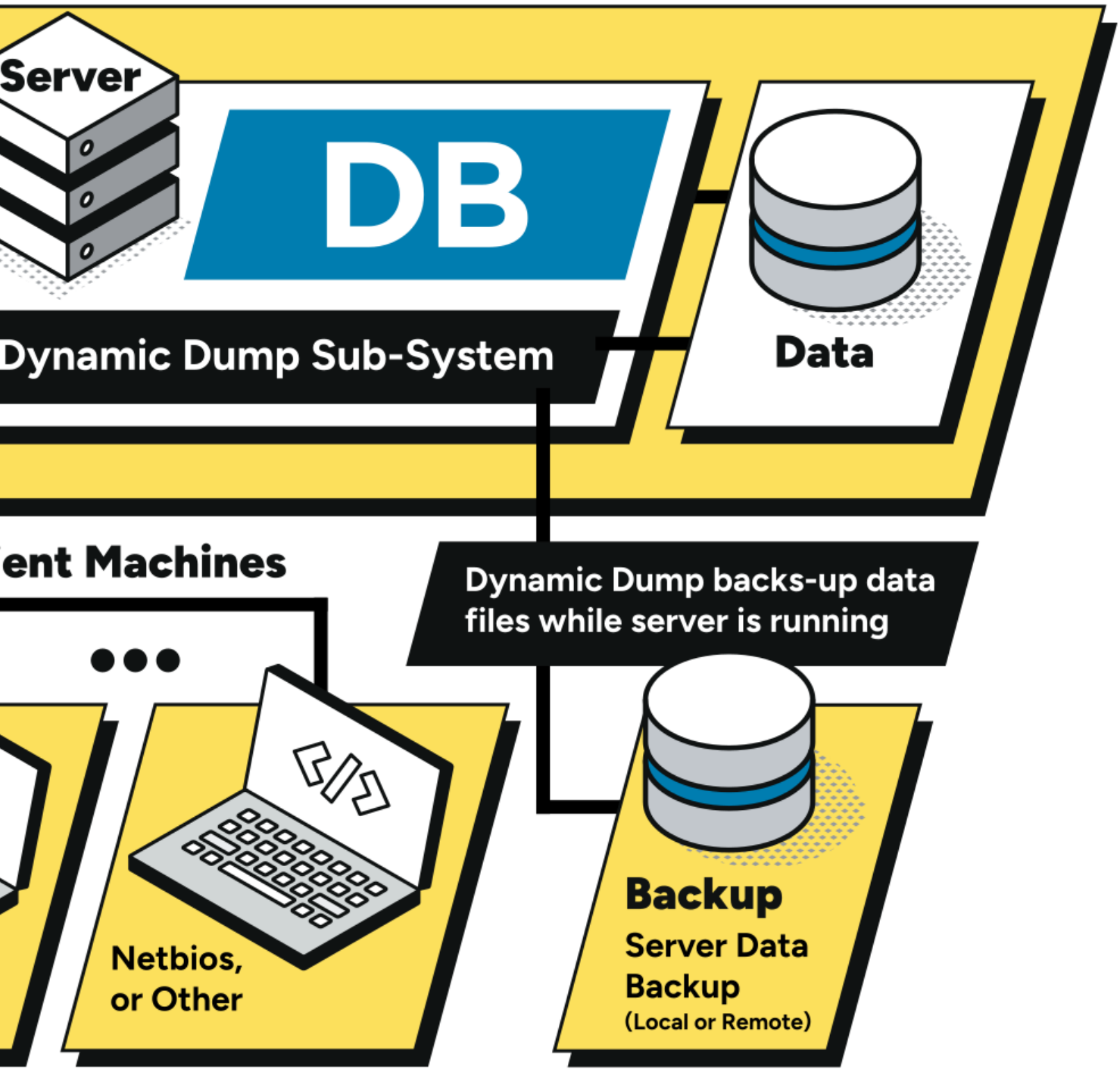
The following call enables VSS writer diagnostic logging:

```
    ctSETCFG(setcfgDIAGNOSTICS, "VSS_WRITER");
```

The following call disables VSS writer diagnostic logging:

```
    ctSETCFG(setcfgDIAGNOSTICS, "~VSS_WRITER");
```

Improvements in Backup/Restore



Additional improvements to the c-treeACE backup technology include:

- **Corrected Dynamic Dump file extensions for non-TRANPROC Files** - The c-treeACE Server failed during a dynamic dump due to a miscalculated size for non-transaction

controlled files being dumped without the !PROTECT option and a non-zero !EXT_SIZE and the total dump size was larger than the first segment. This has been corrected.

- **Mask routine Dynamic Dump messages in Status Log** - Normally a dynamic dump writes the names of all the files it backs up to the c-treeACE Server status log. The CTSTATUS_MASK DYNAMIC_DUMP_FILES keyword can be used to suppress the logging of the names of the files backed up by a dynamic dump operation to reduce the amount of information logged.
- **Corrected ability to kill Dynamic Dump threads** - It was discovered that it was not always possible to terminate a dynamic dump thread waiting for a scheduled dump time. We have resolved this issue.
- **Enhanced logging of Dynamic Dump status messages** - The dynamic dump process can now log the names of the files that it backs up to the SYSLOG file.
- **Files over 4 GB** - Examine Dump Files Utility now displays contents for files over 4 GB.
- **Dynamic Dumps** - Segmented File issues related to Dynamic Dumps have been addressed.

Reduce Performance Impact of Dynamic Dump

When a dynamic dump runs, its disk read and write operations can slow the performance of database operations. The c-treeACE Server now supports an option that allows an administrator to reduce the performance impact of a dynamic dump. The new `DYNAMIC_DUMP_DEFER <milliseconds>` option sets a time in milliseconds that the dynamic dump thread will sleep after each write of a 64KB block of data to the dump backup file.

Mask Routine Backup Messages in CTSTATUS.FCS

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

```
CTSTATUS_MASK DYNAMIC_DUMP_FILES
```

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

Run Time Configuration

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

Examples

To turn the option on:

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

To turn the option off:

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

Enhanced Logging of Dynamic Dump Status Messages

The c-treeACE configuration option `SYSLOG DYNAMIC_DUMP_FILES` configures the dynamic dump process to log the names of the files that it backs up to the *SYSLOG* file. With the addition

of this option, c-treeACE supports the following options for logging the names of files backed up by a dynamic dump:

- To log the names of backed up files to *CTSTATUS.FCS* only (Default):
  No options are needed in *ctsrvr.cfg*.
- To log the names of backed up files to both *CTSTATUS.FCS* and *SYSLOG*:
  Specify `SYSLOG CTSTATUS` in *ctsrvr.cfg*.
- To log the names of backed up files to *SYSLOG* only:
  Specify `CTSTATUS_MASK DYNAMIC_DUMP_FILES` and `SYSLOG DYNAMIC_DUMP_FILES` in *ctsrvr.cfg*.
- To disable logging the names of backed up files:
  Specify `CTSTATUS_MASK DYNAMIC_DUMP_FILES` in *ctsrvr.cfg*.

The **ctSETCFG()** function can be used to turn the masking option on or off while the c-tree Server is operational.

Examples

To turn the option on:

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

To turn the option off:

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

4.9 Index Subsystem

Distinct Key Counts for Duplicate Index

To optimize SQL queries, a more precise estimation of the duplicate index selectivity was implemented, which involves knowing how many distinct keys there are in an index allowing duplicates. Counting distinct partial key values on indexes improves the index selectivity calculations and results in overall improved performance of c-treeACE SQL.

Conditional Indexes

Reference Any Section of the Record

It is now possible to reference any section of the record (offset,length) even if it does not correspond to a field:

It can be very useful to be able to write a filter (or a conditional expression in general) that directly references a section of the record even if this section does not correspond to a field. For conditional expressions, we now allow retrieving a portion of a record as it would be a CT_ARRAY field.

The new function syntax is **FIELD(***offset,size***)** and evaluates as a CT_ARRAY field with length *size* containing the portion of the current record starting at offset *offset*.

Runtime Support Improved to Support IN and BETWEEN Operators

The IN and BETWEEN operators implemented within our conditional expression logic require additional support from FairCom engineers to utilize this power. If you are interested in this capability, please contact FairCom.

Handling of SDAT_ERR (445) When Rebuilding Files

The index rebuild function **RBLIFIL()** now supports deleting records whose key formation fails with error **SDAT_ERR** (445, Not enough data to assemble key value). This situation is not expected; however, it can happen if the data file is damaged. To use this feature, include the constant *purgeIFIL* in the *tfilno* member of the *IFIL* structure you pass to **RBLIFIL()**. This option forces records having illegal duplicate key values to be deleted by the rebuild function (existing behavior) and also records whose key formation fails with **SDAT_ERR** are deleted. The following message is then logged to *CTSTATUS.FCS* for each record that is marked deleted due to error **SDAT_ERR**:

```
Rebuild marked record at offset <offset> deleted due to error 445
```

The index rebuild utility, **ctrbldif**, uses the *-purge* option to enable this behavior.

4.10 Memory Files

Memory Files offer exceptional “in-memory” file and index support resulting in blazing I/O speeds. We have improved this performance even more is this current release.

Concurrency of operations on memory files was improved when adding records. The time spent in holding a temporary file header mutex was reduced to implement this performance improvement.

Also, on 64-bit systems, the c-treeACE hash function that assigns memory file records to memory file hash bins was observed to not evenly distribute memory records among the hash bins for memory address values over 4GB. An alternative hash function has been implemented that improves this distribution on 64-bit systems.

Additionally, the hash computation was also relocated for a slight improvement in performance when more than one client reads/writes a memory file at the same time.

4.11 Segmented Files

Segmented Files allow a single c-treeACE data file to span across multiple hard disk volumes. Although not commonly used in today’s world of large hard disks, this feature comes in very handy on specialized systems.

A number of internal issues have been addressed related to non-transaction processed files, un-buffered I/O, and compacting file segments.

4.12 Super Files

Super Files allow the developer to store multiple c-treeACE data and index files into one single “super” file on disk. Many application developers use this feature to simplify their distribution when an application has a large number of data/index files.

The release addresses several issues related to Super Files in the areas of create, delete, and the new “keep open” technology.

4.13 General File I/O Improvements

The File I/O subsystem has a number of other improvements:

- **Create directories** - We can now automatically create directories that do not exist when creating data and index files.
- **External processes** - On Windows, we now prevent external processes from accessing open files.
- **ctREADFIL** - Improve ctREADFIL sharing modes and introduce new read/write access mode with read-only sharing.
- **File block check** - Improve speed of c-treeACE Server opening files by skipping file block check when no file blocks are active.

- **Shared Reopen** - Permit Shared Reopen after a Transaction Controlled Header is updated.
- **Unlock request** - Ignore the persistent transaction lock state when an unlock request is made outside a transaction.

5. Core Engine Tune-Up

FairCom engineers are constantly at work making sure the c-treeACE core engine—the platform beneath your data and your applications—is working as efficiently and reliably as possible. This section lists changes that give you a core engine you can rely upon.

5.1 Inter-Process Communications - Shared Memory for Unix

The c-treeACE Server and c-treeACE clients on Unix systems now support a shared-memory communication protocol. We support 32-bit clients connecting to a 64-bit c-treeACE Server (and vice-versa) using the shared-memory protocol. We also support shared-memory connections to spin to improve the performance of a single connection. Using the shared-memory communication protocol on Unix has excellent performance results on most systems over TCP/IP.

Shared Memory Client-Server Communication for Unix/Linux

The FairCom Server for Unix supports shared memory connections. Shared memory communication between clients and servers residing on the same machine generally provides much better performance for locally running applications. Local shared memory connections are supported across the board including ISAM and SQL connections, and this includes JDBC and Windows ADO.NET Data providers.

Configuration

Include the following server configuration in *ctsrvr.cfg* to enable this support:

```
COMM_PROTOCOL FSHAREMM
```

FairCom client libraries are compiled with this feature enabled by default.

Note: The `COMM_PROTOCOL` (https://docs.faircom.com/doc/`ctedge_admin/27910.htm`) option specifies the protocol used for ISAM connections. By default, local SQL connections use shared memory unless the `SQL_OPTION NO_SHARED_MEMORY` keyword is specified. See the *c-treeSQL Server Operations and Utilities Guide* for more information about the communication protocol for SQL connections.

System Files, Permissions and Ownership

The FairCom shared memory communication protocol creates a file used by clients to find the shared memory identifier for its shared memory logon region, and creates a Unix domain socket as a file for initial communication between a client and server.

The FairCom Server creates the directory */tmp/ctreedbs* and the file */tmp/ctreedbs/<servername>.logon*. This file name is determined by the value specified with the `SERVER_NAME` configuration option (but see important note below). This file contains an identifier of a shared-memory region used for clients to connect. The following configuration option allows this directory to be directly specified:

```
SHMEM_DIRECTORY <directory_name>
```

IMPORTANT: `SERVER_PORT` applies to the TCP/IP protocol and overrides `SERVER_NAME` if both are used together.

If your server combines shared memory and TCP/IP usage, here are a few tips:

- If you are content with the TCP/IP port resulting from the `SERVER_NAME` option, then use that option and you can connect using the name with either protocol.
- If you wish to explicitly set the TCP/IP port, use `SERVER_PORT` to set that port (then connect with #port to use TCP/IP on that port) and `SERVER_NAME` to set the name used by the shared memory protocol. Note that this approach means that a connection attempt will not be able to 'fall back' to using TCP/IP if the shared memory connection fails, unless you choose your server name so that it matches your `SERVER_PORT` setting. For example, consider the following set of options:

```
SERVER_PORT 7000
SERVER_NAME #7000
```

  Then connect with a server name of #7000. The client will attempt to connect using shared memory first and if that fails it will connect with TCP/IP on port 7000.

Note: on some Linux/Unix systems, be sure to place double quotes around the connection string (for example when using the bash shell): For example: ./ctstat -vas -u admin -p ADMIN -s "#5598ocalhost"
Without the double quotes, bash treats # as a comment, so it ignores the remainder of the connection string.

c-treeACE must have sufficient read, write, create, and delete permissions with this directory. The following server keyword sets the shared memory resource permissions:

```
<permissions>
```

The default is 660. 666 will allow access to c-treeACE by any user account.

Note: Use caution when increasing access permissions to shared memory resources. For example, shared memory permission of 666 allows any user to attach to a shared memory segment and read or write to it. This means that any process can make a request to a FairCom Server or could read the request data of another process through such a shared memory region.

By default, a client application must belong to the server owner's primary group to use shared memory. This is configurable with the SHMEM_GROUP keyword.

```
SHMEM_GROUP  <group>
```

Possible errors indicating problems:

```
FSHAREMM: Could not get group ID for group <group> for shared memory
FSHAREMM: Failed to set group for LQMSG shared memory region: X
```

Shared Memory Keys

When more than one FairCom process is run on a Unix system, the shared memory key used by the servers might hash to the same value, causing problems connecting to the servers. This happens as the **ftok()** system call is used by default to generate the shared memory keys, and **ftok()** is not guaranteed to return unique values. Another possibility is that another unrelated process might happen to use the same shared memory key as generated by the FairCom Server.

An administrator can specify a specific shared memory key for ISAM and SQL shared memory communication protocols to ensure that keys do not match keys already in use on the system. This is specified with the following FairCom configuration options:

```
SHMEM_KEY_ISAM <isam_shared_memory_key>
SHMEM_KEY_SQL <sql_shared_memory_key>
```

The shared memory key values can be specified in either decimal or hexadecimal format. For example:

```
; Set shared memory key for ISAM connections to the specified decimal value:
SHMEM_KEY_ISAM 12345
; Set shared memory key for ISAM connections to the specified hexadecimal value:
SHMEM_KEY_ISAM 0xabcd
```

These server configuration options support specifying an environment variable, whose value is substituted for the configuration option value when the server starts up. For example, if the environment variable MY_ISAM_KEY is set to a numeric value such as 12345 or 0xabcdef before starting the server process, then the following option can be specified in the server configuration file to use this environment variable value for the SHMEM_KEY_ISAM configuration option value:

```
SHMEM_KEY_ISAM %MY_ISAM_KEY%
```

Client Configuration

From the client side, either set the global variable *ctshmemdir* to the directory name before connecting, or set the CTREE_SHMEM_DIRECTORY environment variable. The environment variable takes precedence over the *ctshmemdir* setting. This allows the directory to be dynamically overridden without having to recompile client code.

Errors with Shared Memory Protocol

The FairCom Server logs error messages to *CTSTATUS.FCS* when a shared-memory connection attempt fails. The message is of the form:

```
FSHAREMM: <error message>
```

Adjusting System Limits

When running the FairCom Server with more than 128 shared-memory connections, you may encounter one of the following errors:

```
FSHAREMM: Connect named pipe failure: 13
FSHAREMM: Connect named pipe failure: 28
```

Many Unix/Linux implementations have a default limit of 128 system semaphores, which are used by FairCom shared memory connections. However, this value applies system-wide among all processes.

```
FSHAREMM: Failed to create system semaphore: check system semaphore limits such as SEMMNI
```

The following error can be reported as well:

```
FSHAREMM: Failed to create shared memory segment: check shared memory limits such as SHMMNI
```

These are typically kernel configurations. The FairCom Server requires (`2 + # shared memory CONNECTIONS`) shared memory segments (`SHMMNI`) and semaphores (`SEMMNI`).

The **ipcs** command displays current limits:

```
#ipcs -l
------ Semaphore Limits --------
max number of arrays = 128

------ Shared Memory Limits --------
max number of segments = 128
```

To increase limits to allow up to 1024 shared memory segments and semaphores, consider adding the following to your local */etc/sysctl.conf* file.

```
kernel.shmmni = 1024
kernel.sem = 250 256000 32 1024
```

Run this command to then enable support:

```
/sbin/sysctl -p
```

Note: In general, you will require root superuser access to make these changes. Consult your specific Unix/Linux documentation for the actual file location and parameters of this configuration.

Usage

To take advantage of this feature, check the following:

1. Shut down your FairCom Server and add the following keyword to your *ctsrvr.cfg* file: `COMM_PROTOCOL FSHAREMM`
2. Restart the FairCom Server.
3. Execute any FairCom utility you've linked with V9.5 or later of the FairCom Server you have on the same machine as the FairCom Server process. Even if you are linked with a c-tree TCP/IP library, it will automatically detect if you are running on the same machine and try to

connect via shared memory. This way you don't need multiple versions of your application and utilities.

4. You can monitor your connections by listing the clients from the **ctadmn** command-line utility on Linux, or by using the c-treeACE Monitor program from Windows (it is shown in the **Comm Info** column on the far right).

Usage Notes

- Unix and Windows client libraries are built with shared-memory support by default.
- When the FairCom Server detects a request to connect from the same machine as the client, it first attempts to connect using shared memory. If that succeeds, the connection uses the shared-memory protocol. Otherwise, the connection is made using TCP/IP.
- 32-bit clients can connect to 64-bit servers (and vice versa).
- By default, a client application must belong to the server owner's primary group to use shared memory. This is configurable with the `SHMEM_GROUP` keyword.
- Shared memory uses a Unix domain (file system) socket for transferring data between the client and server. The Unix domain socket exists as a file named `/tmp/ctreedbs/<servername>.logon`.

  The `COMPATIBILITY SHMEM_PIPE` option has been deprecated and no longer has any effect.

- A Unix/Linux server using shared-memory communications will create a directory */tmp/ctreedbs*. If this directory already exists (for example, if a different user had started the server, even from a previous run) and the server does not have write permission to this directory, startup will fail, most likely reporting a **DSRV_ERR** error (509, duplicate server), even if no other server is currently running.
- **pthread** mutex shared-memory support is expected for Unix systems. However, if a client and the server are compiled with incompatible options (for example, the client uses System V semaphores but the server uses **pthread** mutexes) the connection attempt will fail and the FairCom Server will log one of the following messages to *CTSTATUS.FCS*:

  If client is using System V semaphore and server is using **pthread** mutex:

```
A shared-memory connection attempt by an incompatible client failed: pthread mutex required
```

  If server is using System V semaphore and client is using **pthread** mutex:

```
A shared-memory connection attempt by an incompatible client failed: SYSV sema required
```

System Tools

The Unix/Linux **ipcs** utility is useful for listing the shared-memory regions and semaphore sets that currently exist on a system. **ipcs** can also be used to remove shared-memory regions and semaphore sets. Under normal circumstances, the FairCom Server removes shared-memory regions and semaphore sets for connections that have been closed. However, if the FairCom Server process terminates abnormally, it may be necessary to manually remove the shared-memory regions and semaphore sets belonging to this process.

5.2 Auto-Numbering Support

IDENTITY Column Auto-Incrementing Support

A new numeric column attribute for auto-incrementing columns is now available. This feature is enabled with the `IDENTITY` SQL keyword.

Syntax

```
exact_numeric_data_type ::
     TINYINT
|    SMALLINT
|    INTEGER
|    BIGINT
|    NUMERIC
|    NUMBER [ ( precision [ , scale ] ) ]
|    DECIMAL [(precision, scale)]
|    MONEY [(precision)]
[ IDENTITY [ ( ± seed , ± increment ) ] ]
```

- **IDENTITY**

  For `TINYINT`, `SMALLINT`, `INTEGER` and `BIGINT` column types, an optional auto-incrementing attribute can be defined with the `IDENTITY` option. This adds the column of the defined type to the table and automatically updates the value on each row insert.

  The `IDENTITY` attribute by itself does not guarantee uniqueness of assigned values. Use a unique index to ensure unique values.

  `IDENTITY` can optionally specify *seed* and *increment* values. *seed* is the starting assignment value and is incremented by *increment* for each update.

```
CREATE TABLE t1 (name CHAR(10), id_num INTEGER IDENTITY (0, 1));
```

  Only one `IDENTITY` column can be defined per table. `IDENTITY` columns cannot be specified on tables with only one column.

  `IDENTITY` values assigned to aborted rows in a table are lost. Note that this can result in gaps in the numerical sequence order.

  `IDENTITY` is not supported for `NUMERIC`, `NUMBER`, `DECIMAL` or `MONEY` column types.

  `IDENTITY` cannot be added to an existing field via ALTER TABLE.

To return the current IDENTITY value in effect, the scalar function **LAST_IDENT()** can be called.

IDENTITY Support Added to FairCom DB API

The following functions were added to the FairCom DB API C API for working with Identity fields.

ctdbSetIdentityField

Set an Identity field for a table.

```
CTDBRET ctdbSetIdentityField(CTHANDLE Handle, pTEXT FieldName, CTINT64 seed, CTINT64 increment)
```

ctdbSetIdentityField() returns CTDBRET_OK on success or a c-tree error code on failure.

Note: You must call **ctdbAlterTable()** to persist the change to the table after this call.

ctdbGetIdentityFieldDetails

Retrieve the name, seed and increment of an Identity field.

```
pTEXT ctdbGetIdentityFieldDetails(CTHANDLE Handle, pLONG8 seed, pLONG8 increment)
```

ctdbGetIdentityFieldDetails() returns the Identity field name or NULL. For a NULL return, use **ctdbGetError()** to verify if there was an actual error or if there is no identity field defined.

ctdbGetLastIdentity

Retrieve the last Identity value used for a table.

```
CTDBRET ctdbGetLastIdentity(CTHANDLE Handle, pLONG8 value)
```

ctdbGetLastIdentity() returns CTDBRET_OK on success or a c-tree error code on failure.

.NET Support

- **CTTable.SetIdentityField(***table, seed, increment***)**
- **CTTable.GetIdentityFieldDetails(***seed, increment***)**
- **CTTable.GetIdentityField()**
- **CTTable.GetLastIdentity()**

IDENTITY Columns Imported with the c-treeACE SQL Import Utility

c-treeACE SQL V9.3 introduced a new IDENTITY attribute for exact numeric columns. This support can be enabled on data files at the ISAM layer. The c-treeACE SQL Import utility has been enhanced to detect this attribute in existing tables and update the corresponding SQL system tables upon import.

Sequence Numbers

Sequential numbering is a requirement for many applications. Accounting data is a particularly heavy application usage with respect to invoices, quotations, and other receivables and payables documentation. c-treeACE offers several forms of automated numbering. The serial segment (*SRLSEG*) feature has long served this purpose. However, it suffers some limitations as it is an additional index overhead on a per-file basis. *IDENTITY* support is more flexible and performs better, although it is still limited to a single table. Consider global operations against multiple servers that require a single numbering scheme. This requires a persisted pool, and indeed, many c-treeACE applications have implemented their own unique solutions.

c-treeACE V10 introduces a generic method to create, maintain, and access multiple sources of sequential numbers through a persisted **Sequence** feature.

c-treeACE Sequence Number Store

A sequence has the following properties associated with it:

- sequence name
- initial value
- current value

- limit value
- incrementing or decrementing sequence
- increment or decrement amount
- cycling or terminating sequence

Changes are immediately committed to the sequence number. This includes creation and deletion of the sequence, as well as changing sequence settings and current values. A complete API is provided to manage this sequence number store.

Attributes for each sequence are stored in a fixed-length record in the files *SEQUENCEDT.FCS* and *SEQUENCEIX.FCS*. c-treeACE creates these files when starting if they do not exist and these files are under full transaction control. An index on the sequence name field is used to find the record. Sequence values are stored as 64-bit values as shown in the *ctSEQATTR* structure definition.

File Resource Fork - Direct Access Resource (DAR)

c-treeACE has long supported an ability to store “resource” records in a file. This resembles a similar approach taken by Apple Macintosh files that originally included both a “data” fork and a “resource” fork.

Resource forks store non-volatile data. Some people think of them as an “extended header” of a file. c-treeACE has supported “resource” forks for records to store various types of information including data schemas. However, in contrast to the main file header, these resources are not designed for high-speed access. A new design was required for fast access and updates.

To this end, a new special-purpose resource was introduced: the Direct Access Resource (DAR). The c-treeACE IDENTITY feature (auto-increment fields) takes advantage of this new feature and experiences performance similar to the primary header portion of the file. This performance was not possible with the prior existing resource or *SRLSEG* index features.

If you encounter a situation where you need to store metadata in the resource portion of a file and require high-speed access, check out the new DAR support.

Note: Files created with c-treeACE post V9.2 include this DAR resource by default. Older applications and utilities may not always recognize this new resource.

5.3 Thread Impersonate - Connection Impersonating Another Connection

A unique feature we call “thread impersonate” has been implemented: We have introduced an API that can be used to cause a connection to impersonate another connection.

Note: This advanced feature should only be used by developers who have an in-depth knowledge of the way c-treeACE uses threading in your operating system.

The c-treeACE Server maintains a separate thread for each connected client. Each server thread has its own internal “Owner ID” to maintain context information associated with that

user/connection. The server listens on two separate communications ports: one for SQL connections and one for ISAM connections. When a SQL or ISAM client connects, it has its own server thread/context.

We received customer requests to allow both the SQL and ISAM operations to be coordinated under a single database transaction. A new thread impersonation feature brings an advanced level of connection control never possible before.

A user can now connect a SQL client and an ISAM client, and then indicate to the server that the ISAM client wants to impersonate the SQL client. On the server, both connections will use the same “Owner ID” appearing as a single thread/context. As a result, sophisticated users can intermingle SQL and ISAM code within the same code/database transaction. This is a powerful method to gain performance advantages, particularly in instances where a speedy ISAM call is preferable to a slower SQL query.

Thread Impersonation is an advanced feature and requires careful implementation so contact your nearest FairCom office if you have interest in this technology.

Several API functions have been added to enable this feature.

FairCom Low-Level

- **ctImpersonateTask( )**

FairCom DB API C API

- **ctdbBeginImpersonation( )**
- **ctdbEndImpersonation( )**

FairCom DB API C++ API

- **CTSession::BeginImpersonation( )**
- **CTSession::EndImpersonation( )**

c-treeACE SQL

- **fc_get_taskid( )**
- **fc_set_impersonation( )**

This section describes the functions used to implement thread impersonation:

• **ctImpersonateTask** (page 197)

• **ctdbBeginImpersonation** (page 200)

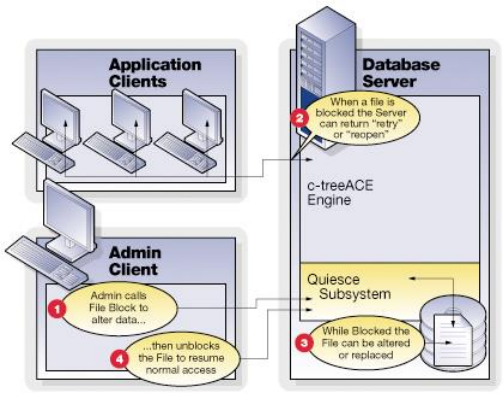
• **ctdbEndImpersonation** (page 201)

• **CTSession::BeginImpersonation** (page 202)

• **CTSession::EndImpersonation** (page 203)

5.4 Quiesce

The powerful ability to suspend (quiet or quiesce) c-treeACE Server's operations and later re-enable them allows administrators to perform maintenance or other on-demand activities without having to stop an application. Users are temporarily held back from operations. Files can be readily accessed for backup, especially useful for hardware-based disk snapshot utilities.



We now allow Non-ADMIN Users to Quiet c-treeACE. The c-treeACE quiesce feature is typically only enabled for an ADMIN user; however, there are times when the administrator may allow a non-ADMIN user to have this ability. To allow a non-ADMIN user to quiesce the c-treeACE Server via a call to **ctQUIET()**, the server configuration keyword COMPATIBILITY NONADMIN_QUIET has been added. This parallels the functionality of the existing COMPATIBILITY NONADMIN_FILBLK feature.

5.5 Startup/Shutdown

Enhancements related to server startup and shutdown include:

- **Enhanced Windows Service Shutdown Operations** - Windows Vista and later allow a new service PRESHUTDOWN notification. With normal SHUTDOWN notification, the service will be killed if it takes more than approximately 90 seconds to stop. When a service signals that it handles a PRESHUTDOWN notification, it will be given a greater amount of time to complete its shutdown. This can avoid a lengthy auto-recovery should the server have a large number of cache pages to flush at shutdown, such that it cannot complete under the SHUTDOWN limit. c-treeACE now enables handling of the PRESHUTDOWN notification on the selected Windows versions.
- **Faster Server Shutdown with Large Memory Allocations** - The c-treeACE database engine normally frees associated memory (data and index caches) when shutting down if all clients have disconnected. However, it was discovered that freeing this memory can take a long time when using very large data and index caches due to the overhead of returning the allocated memory blocks to the process heap. When the c-treeACE process is about to exit, returning the memory to the process heap is not necessary. This housekeeping activity is now skipped when shutting down to avoid observed delays.

5.6 Other Tune-Ups

A number of other minor additions include:

- **Include server name and version in c-treeACE Server system tray text** - The c-treeACE Server for Windows now displays its version, server name, SQL port, and SQL database as the tool tip for the system tray icon.

- **c-treeACE Server for Windows now accepts command-line options when run as a service** - A customer wanted to specify the location of the c-treeACE Server configuration file when running the c-treeACE Server as a Windows service. But they found that when the c-treeACE Server is run as a Windows service, it ignores the command-line options. We modified the server's startup code so that it gets the service's command-line options from two locations and processes them. The two locations are: a) the options specified in the ImagePath registry key (which are passed to the **WinMain()** function), and b) the options specified in the "Start parameters" text box if using the Windows services control panel applet to start the service.
- **COMMIT_DELAY 2 for Windows systems** - On Windows our experience shows that using COMMIT_DELAY can dramatically improve performance of TRNLOG transactions. We set now default to COMMIT_DELAY 2 for Windows systems.
- **Insufficient disk space** - A new c-treeACE Server configuration option monitors available disk space and shuts down database engine when insufficient disk space is available. This configuration option DISK_FULL_ACTION enables c-treeACE Server to monitor available disk space and to shut down the database engine when the disk space falls below the specified limit.
- **Flush of file system cache can now be disabled for non-transaction files** - While testing on Windows, we noticed time spent in our flush file buffers logic. We call this logic on the first update to a file after it is opened in order to write the header with the update flag value set, so that if c-treeACE terminates abnormally, the next time we open the file we see the update flag set and know that the file must be processed by automatic recovery (or rebuilt if it is a non-transaction file). For a temporary non-transaction-controlled file, it might be desirable to avoid this flush logic. We now provide a way to suppress this call for a non-transaction-controlled file.
- **Increase default TRANSACTION_FLUSH setting** - Testing automatic recovery performance justified that we increased the default TRANSACTION_FLUSH setting from 100 to 500000 and changed the TRANSACTION_FLUSH setting in the configuration that we include with the c-treeACE Server package from 10000 to 500000. The result was an increased performance by reducing flushing of data and index cache pages.
- **Exported C++ Methods in the Server .DLL Model** - The c-treeACE Server .dll model is a power extension of c-treeACE database technology allowing all of the features of c-treeACE Server technology to now be available to your existing non-server applications. This is accomplished by having the core server bundled into a run-time linkable shared library object. Initially, this model excluded C++ methods, as the methods were not exported from the library. These exports have since been added.
- **Maximum Virtual Files (MAXVFIL) changed from 500 to 32767** - We decided that because of today's modern operating systems, that the default value for MAXVFIL should be increased, therefore we increased the value from 500 to 32767. This also resolved an issue within Dynamic Dumps while a large amount of files were in operation.
- **CPU_AFFINITY option now ignores invalid CPU identifiers on Windows systems** - On Windows systems, the c-treeACE Server's CPU_AFFINITY option now ignores any invalid CPU identifiers that are specified. For example, the following CPU_AFFINITY option is now valid on a 2 CPU system: CPU_AFFINITY 0,2,3. In this example CPU identifiers 2 and 3 are ignored and the c-treeACE Server process is set to run only on CPU 0.
- **HP/UX Itanium system** - c-treeACE Server showed high CPU usage on HP/UX Itanium system when activated with time-limited evaluation key. We fixed an internal security check

within the server code that was “polling” and consuming CPU resources. This only occurred if an evaluation activation key was used for the server, and this logic only pertains to the security checks for evaluation servers.

Enhanced c-treeACE Monitoring of Full Disk Conditions

A full disk can result in a critical condition for c-treeACE. As the ability to log transactions and write to transaction controlled files is a fundamental necessity, it is important to avoid low disk space conditions. Previously, c-treeACE provided options (DISK_FULL_LIMIT and DISK_FULL_VOLUME) to check for disk space, however, these configurations only made the check after a file extension had completed, and did not provide for a proactive solution. To address this concern with a robust proactive approach, an extended option is available to allow for specified actions to take place when a low disk space condition is detected.

The c-treeACE configuration option `DISK_FULL_ACTION` enables c-treeACE to monitor available disk space and to shut down the database engine when disk space falls below a specified limit. The usage is:

```
DISK_FULL_ACTION <action> <volume> <limit>
```

Currently, the only supported value for *<action>* is *shutdown*. This action enables c-treeACE to shut down when the specified disk full limit is reached.

If the path contains spaces, they must be surrounded with double quotes as shown in the following examples:

Windows examples:

```
DISK_FULL_ACTION     shutdown D:\                                          500 MB
DISK_FULL_ACTION     shutdown "C:\Users\Administrator\My Documents"   1 GB
```

Unix examples:

```
DISK_FULL_ACTION     shutdown /users/administrator                  500 MB
DISK_FULL_ACTION     shutdown "/users/administrator/my documents"     1 GB
```

The keyword can be specified multiple times with actions specified on a per volume basis. `DISK_FULL_ACTION` is also independent of the `DISK_FULL_LIMIT` and `DISK_FULL_VOLUME` keywords. (These older keywords result in disk writes to fail when the limit is reached.)

When this keyword is specified, c-treeACE creates an administrative thread that checks the disk space conditions every 60 seconds. It takes the specified *action* (currently only shutdown supported) if the limit is reached.

The following new messages appear in *CTSTATUS.FCS* when this option is used:

1) c-treeACE logs a message for each `DISK_FULL_ACTION` option it finds in *ctsrvr.cfg*:

```
 - User# 00001       Configuration info : DISK_FULL_ACTION threshold set for volume S:\ to 7516192768 bytes
```

2) c-treeACE logs a message to indicate that the available disk space monitoring thread has started:

```
 - User# 00009       Thread Start: ctDISKFULLactionthrd
```

3) c-treeACE logs a message when it detects disk space below the specified threshold which causes the database engine to shut down:

```
 - User# 00009      DISK_FULL_ACTION: Space on volume S:\ is 5338173440, which is below threshold of
7516192768. Initiating database engine shutdown.
```

c-treeACE on Windows now Prevents External Processes from Accessing Open Files

For c-tree data and index files that do not use the *DUPCHANEL* filemode, c-treeACE for Windows now opens files in shared read-only mode such that only c-treeACE can write to the files.

The c-tree error code, **FNAC_ERR** (920), is now returned when a file exists but is not accessible, for example, due to file system permission settings or file sharing restrictions. If another process has a c-tree data or index file open for write access and c-treeACE fails to open the file, the file open fails with error 920, and the *sysiocod* is set to the system error code.

For a c-treeACE SQL client, error -17920 corresponds to a c-tree error 920. For example, if a process opens a data file with write access and then isql attempts to open the table, the c-treeACE SQL logs the following status log entry:

```
error(-17920): CT - The file exists but could not be accessed: check for permission or sharing restrictions
```

Important Notes

- This feature is only available on Windows.
- A process can still write to a c-tree data or index file that is not currently open by c-treeACE. This includes files that use the *ctVIRTUAL* filemode if c-treeACE has “virtually closed” the file, that is, if it has closed the file descriptor for that file even though the file remains logically open.
- If a process has the file open for write access and c-treeACE attempts to open the file for write access (and read-only sharing), c-treeACE will fail to open the file. Previously, c-treeACE was able to open the file in this situation.
- There are two types of file opens that are considered:
  - Standard physical file opens (in which the file descriptor is stored in the system file control block list). In this case the file is opened for read/write access and read-only sharing is permitted.
  - Special file opens (for example, when replication opens a transaction log, or when the c-tree Server opens a file to check if it already has the file open). In this case the file is opened for read-only access and read/write sharing is permitted. This is necessary as c-treeACE may already have the file open for write access.

A configuration option, `COMPATIBILITY OPEN_SHARE_RW`, is available to restore the previous behavior of opening the files with read/write share access.

Exported C++ Methods in the Server .DLL Model

The c-treeACE Server .DLL model is a power extension of c-treeACE database technology allowing all of the features of c-treeACE Server technology to now be available within your existing non-server applications. This is accomplished by having the core server bundled into a run-time linkable shared library object.

Initially, this model excluded C++ methods, as the methods were not exported from the library. These exports have since been added.

FairCom DB API Functions and Methods Not Supported with ctFILBLK and ctQUIET

FairCom DB API C functions and C++ methods are handled differently than standard C function calls. Specifically, FairCom DB API calls do not route through the internal c-treeACE Server "Single Entry Point" or SEP control handling. SEP allows advanced control over some server functions, most notably, the file block and server quiesce features. As a consequence, these features are not supported when using the c-treeACE Server .DLL model. Calling any FairCom DB API function or method while the server .DLL is in a blocked state will result in an unpredictable manner, and will most likely cause a process fault and terminate. Indeed, any shared object library faults will terminate the process that loaded them.

Process Stack Dumps Enabled for All Unix Servers

In the extremely rare event that an internal c-treeACE error should occur which necessitates halting the server process, c-treeACE now calls the **pstack** utility on all Unix systems. This results in a core file and call stack to be dumped, when at all possible, for problem resolution by the FairCom engineering team. This dump captures the exact working environment at the time of failure and can be quickly analyzed.

Should such a failure occur, be prepared to send the resulting core file and any generated **pstack** files to your FairCom support team.

Note: "In-flight" data will be captured at the time these core files are dumped. Please consult with your local risk assessment organization concerning any privacy or security issues before forwarding this information to FairCom.

6. Security Lockdown

FairCom conducts ongoing security reviews, resulting in numerous elevated security controls required to safeguard your mission-critical data. This section lists the important advances that have been made to "lock down" your security.

6.1 Key Store Support for Advanced Encryption

c-treeACE advanced encryption (AES, Blowfish, Twofish, 3DES) requires a master password to protect encrypted file access. By default, this master key must be presented to c-treeACE on startup as prompted. However, this prompted interaction is not always possible. Consider the case of a failover strategy for business continuity, or the case where no single person should ever know the complete key as keys are built from random secure key generators. c-treeACE now supports a key store to obtain this key value at startup.

The **ctcpvf** utility supports command-line options that can be used to select the master key cipher strength, and to write the master key to an encrypted file:

```
usage: ctcpvf  [-c <cipher>] [-f <filename>] [-k <key>] [-s <store>]
  -c <cipher>          use encryption cipher <cipher>
                         Supported ciphers: aes256 aes128
                         Default is aes256
  -f <filename>        create password verification file <filename>
                         Default is ctsrvr.pvf
  -k <key>             use <key> as the master key
  -s [<store>]         store key in encrypted file <store>
                         Default is ctsrvr.fkf
```

The c-treeACE configuration option `MASTER_KEY_FILE` specifies a file from which c-treeACE reads the master encryption key. On Linux and Unix systems, the master key is stored AES encrypted in a file on disk, with permissions set such that only the user that created the file can

read it (permissions are set to 400). For complete security, it is important to use hardened safeguards to fully protect this key store file.

Note: The key file (or user key on Linux and Unix) is encrypted using AES. The encryption is intended to only prevent casual inspection of the data when the file's contents are viewed. The permissions on the file are the defense against an unauthorized user reading the file. The Windows master key approach uses the DPAPI to encrypt the data using the user credentials, and only that user can then decrypt the file. Unix support is a bit weaker because it relies on the file permissions, which could potentially be changed such that another user could read and decrypt the key.

6.2 Tightened Access Security Controls

c-treeACE has many advanced security controls to prevent unauthorized access to data. Multiple areas of access control were reviewed and updated to take advantage of the latest security techniques where applicable. Some areas, such as password transmission and storage, were extensively modified in V10, so please note the compatibility issues that have been introduced.

- The *FAIRCOM.FCS* file stores user and group information required to access c-treeACE. This file is now encrypted by default (the `ADMIN_ENCRYPT` keyword defaults to YES). To disable this encryption level, add `ADMIN_ENCRYPT NO` to your *ctsrvr.cfg* configuration file. By default, the file is scrambled using FairCom's CAMO technique. (CAMO or "Camouflage" is an older, legacy method of hiding data, which is not a standards-conforming encryption scheme, such as AES. It is not intended as a replacement for Advanced Encryption or other security systems.) AES encryption is used if `ADVANCED_ENCRYPTION YES` is specified in *ctsrvr.cfg*.
- A GUEST account is used when no user name is presented to c-treeACE at connection time. Guest logons are now disabled by default. Add `GUEST_LOGON YES` to your *ctsrvr.cfg* configuration file should you wish to continue using the guest account.
- Strong passwords are the first line of defense to protect unauthorized access to data. c-treeACE has increased password lengths to 63 bytes. Passwords are now "salted" and stored as SHA-512 hash values. Enhanced password exchange between client and server has also been implemented. Password exchanges with the server that must send a new password (for example, the **ctpass** utility) use AES encryption of the data between the client and server with randomly negotiated keys. Collectively, these techniques provide advanced levels of password protection.

  These password changes break prior client/server compatibility with existing clients and/or servers. Only V10 clients matching the new V10 server format can connect to c-treeACE V10. New error codes can be experienced when a mismatch between client and server occur with these new secure logon changes.
  - If a client without secure password transmission calls a function that sends a password and the server requires the new secure logon, the call fails with error **PWDC_ERR** (961).
  - If a client with secure password transmission calls a function that sends a password and the server does not support secure password transmission, the call fails with error **PWDS_ERR** (962).

There is no reverse compatibility configuration available ensuring secured password exchange going forward.

See also **V10 Client/Server Compatiblity** (page 214).

6.3 Restricted Security Administrator Access

Prior to this enhancement, c-treeACE allowed any user to connect using the **ctadmn** utility and change the password for any user account. **ctadmn** used low-level and ISAM-level functions to read and write records in the c-treeACE Server's security files (which are members of the *FAIRCOM.FCS* superfile).

Tighter administrator security restrictions have been enabled such that c-treeACE now permits *FAIRCOM.FCS* to be opened only by a member of the ADMIN group. ADMIN group members can read data from *FAIRCOM.FCS* using low-level or ISAM function calls but cannot update *FAIRCOM.FCS*, other than through calls to the **SECURITY()** API function.

The new security restrictions are as follows:

- **Add user account** - Only members of the ADMIN group can add user accounts, and only super ADMIN can add a user account as a member of the ADMIN group.
- **Delete user account** - Only members of the ADMIN group can delete user accounts, and only super ADMIN can delete a user account that is a member of the ADMIN group.
- **Add a user to a group** - Only members of the ADMIN group can add a user to a group, and only super ADMIN can add a user to the ADMIN group.
- **Remove a user from a group** - Only members of the ADMIN group can remove a user from a group, and only super ADMIN can remove a user from the ADMIN group.
- **Change user description or password** - Members of the ADMIN group can change the description or password of non-ADMIN group users, and any user can change his own description or password.
- **List user accounts** - Only members of the ADMIN group can list user accounts.
- **Show user account information** - Only members of the ADMIN group can show account information (for any user).
- **Change user memory limit** - Only members of the ADMIN group can change memory limit, and only super ADMIN can change memory limit for a member of the ADMIN group.
- **Change user extended settings** - Only members of the ADMIN group can change extended settings, and only super ADMIN can change extended settings for a member of the ADMIN group.
- **Add a group** - Only members of the ADMIN group can add groups.
- **Remove a group** - Only members of the ADMIN group can remove groups.
- **List groups** - Only members of the ADMIN group can list groups.
- **Show group information** - Only members of the ADMIN group can show group information.
- **Change group description** - Members of the ADMIN group can change the description of any group.
- **Change group memory limit** - Members of the ADMIN group can change the memory limit of any group.
- **List files matching filename** - All users can list files matching the specified filename

- **Change file group** - All non-ADMIN users can change the group of a file that they own. ADMIN group users can change the group of any file.
- **Change file owner** - All non-ADMIN users can change the owner of a file that they own. ADMIN group users can change the owner of any file.
- **Change file password** - All non-ADMIN users can change the password of a file that they own. ADMIN group users can change the password of any file.
- **Change file permissions** - All non-ADMIN users can change the permissions of a file that they own. ADMIN group users can change the permissions of any file.

Compatibility

If a c-treeACE client library or utility (for example, **ctadmn** or **sa_admin**) that does not have these security enhancements enabled attempts to perform security operations on a c-treeACE Server that enforces the tighter security restrictions, the operation might fail with one of the following error codes, even if the c-treeACE Server allows that user to perform the specified operation:

- **SACS_ERR** (456) Group access denied
- **SWRT_ERR** (458) Write permission not granted

These errors may occur because the client logic prior to this enhancement may attempt to open *FAIRCOM.FCS* or read or write to *FAIRCOM.FCS* using low-level or ISAM client-side function calls. The client-side logic that is compatible with the new security restrictions avoids these errors by using the new **SECURITY()** API function modes to perform these operations.

If a c-treeACE client library or utility that has these security enhancements enabled attempts to perform security operations on a c-treeACE Server that does not enforce the tighter security restrictions, the operation may fail with the following error, because the c-treeACE Server does not support the **SECURITY()** API modes that were added with this enhancement:

- **SCMP_ERR** (925) The c-treeACE client is attempting to use features of the **SECURITY()** API function that this c-treeACE Server does not support. Update your c-treeACE Server.

An additional error code was added indicating a security operation failed because it can only be performed by the super ADMIN user account:

- **SADM_ERR** (924) Only the super administrator user account (named ADMIN) can perform this operation. For example, only ADMIN can change ADMIN group membership.

If *FAIRCOM.FCS* exists when c-treeACE starts, the server automatically makes any necessary changes to the file and its members to enforce these tighter security restrictions. If *FAIRCOM.FCS* does not exist, the server creates the file with all appropriate security attributes.

These security changes within *FAIRCOM.FCS* include:

- Set file permissions on *FAIRCOM.FCS* and its members so that only the super ADMIN account can write to these files; ADMIN group members can read from these files; and other users can neither read nor write these files.
- Set a file attribute so that no client can write to the *FAIRCOM.FCS* member files using low-level or ISAM function calls. Only the **SECURITY()** function can write to these files using the new modes that were added in this revision.
- The server adds *DODA*s to the *FAIRCOM.FCS* data file members so that the client properly reads security records when running in a heterogeneous environment.

Security API Modes

The following modes have been added to the **SECURITY()** API function to support the new security restrictions:

Symbolic	Value	Description
SEC_ADD_USER	13	Add user account
SEC_REMOVE_USER	14	Remove user account
SEC_CHANGE_USER_GROUPS	15	Change user group membership
SEC_CHANGE_USER_DESC	16	Change user description
SEC_CHANGE_USER_PASSWD	17	Change user password
SEC_CHANGE_USER_MEMORY	18	Change user memory limit
SEC_CHANGE_USER_XINFO	19	Change user extended settings
SEC_ADD_GROUP	20	Add group
SEC_REMOVE_GROUP	21	Remove group
SEC_CHANGE_GROUP_DESC	22	Change group description
SEC_CHANGE_GROUP_MEMORY	23	Change group memory limit

Example

See the *ctadmn.c* source module, which demonstrates specifying various parameter values in the input buffer when calling the **SECURITY()** API function with these new modes.

6.4 User and Group Logon Limits

c-treeACE Server now supports limiting users to a specified number of concurrent logons based on their user name and/or group membership. These limits are stored in *FAIRCOM.FCS* and can be set using the **ctadmn** utility. By default, these limits are zero, meaning no limit.

The **ctadmn** utility's User Operations menu supports a new option: Change User Logon Limit, and the Group Definitions menu supports a new option: Change Group Logon Limit.

The **SECURITY()** API function supports two new modes: SEC_CHANGE_USER_LOGLMT and SEC_CHANGE_GROUP_LOGLMT.

The **sa_admin** utility was not updated to support options to set these limits.

When logon limits are in effect for a user account or one of its groups, an ISAM or SQL connection attempt can fail with one of the following errors:

- **ULMT_ERR** (967, Logon is denied because this user account has reached its maximum number of concurrent logons.)
- **GLMT_ERR** (968, Logon is denied because one of the groups for this user account has reached its maximum number of concurrent logons.)

6.5 Server Utilities Updated for Use with Advanced Encryption

The **ctrdmp** restore and **ctfdmp** (forward roll) utilities now detect when transaction logs are using advanced encryption and when data or index files that are referenced in the transaction logs use advanced encryption. When the transaction logs use advanced encryption, these utilities prompt for the master password after opening the first transaction log, which is essentially at the beginning of the restore or forward roll process. When data or index files referenced in the logs use advanced encryption, the user is prompted for the master password the first time it finds a reference to a file that uses advanced encryption. This requires the *ctsrvr.pvf* password file to be created and present.

The **ctdmpidx** dump index and **ctvfyidx** index verify utilities now detect when the index file being inspected is encrypted with advanced encryption. In this case, they turn on advanced encryption support and prompt for the master encryption password just as the server does at startup.

6.6 Other Lockdowns

c-treeACE services thousands of mission-critical applications in many sensitive industries including banking, finance, and health care. The increasing demand to provide users with highly secured access to systems and data continues to impact developers in these and many other industries. Providing the necessary security required to safeguard mission-critical data is crucial.

In our ongoing security reviews, we have implemented numerous elevated security controls assisting developers in securely transmitting and storing data:

- **Enhanced security of c-treeACE logons** - Secure logon for ISAM and SQL connections in which the client never passes any user's password in a form that can be deciphered.
- **Server encryption master key enhancement** - Increased advanced encryption master key length from 128 to 256 bits and, on Windows, added support for reading the master key from an encrypted file when c-treeACE Server starts. c-treeACE's advanced encryption uses a master encryption key to encrypt the following items using the AES cipher:

  a) the security resource in c-treeACE data and index files that use advanced encryption,
  b) the encryption key in the transaction log file header when using advanced log encryption.

  Now, when advanced encryption and *FAIRCOM.FCS* encryption (ADMIN_ENCRYPT) are enabled, when the *FAIRCOM.FCS* file is created it is encrypted using AES 256-bit encryption. When using advanced encryption and transaction log encryption, c-treeACE now uses a 256-bit key to encrypt the transaction logs. (Previously, the transaction log encryption key was 128 bits.)
- **Function to change master encryption password** - The advanced encryption feature uses a master password to encrypt the file-specific advanced encryption key in data, index, and transaction log files that use advanced encryption. A new function can change the password used to encrypt the file-specific encryption keys in the specified files.

7. Interface Developments

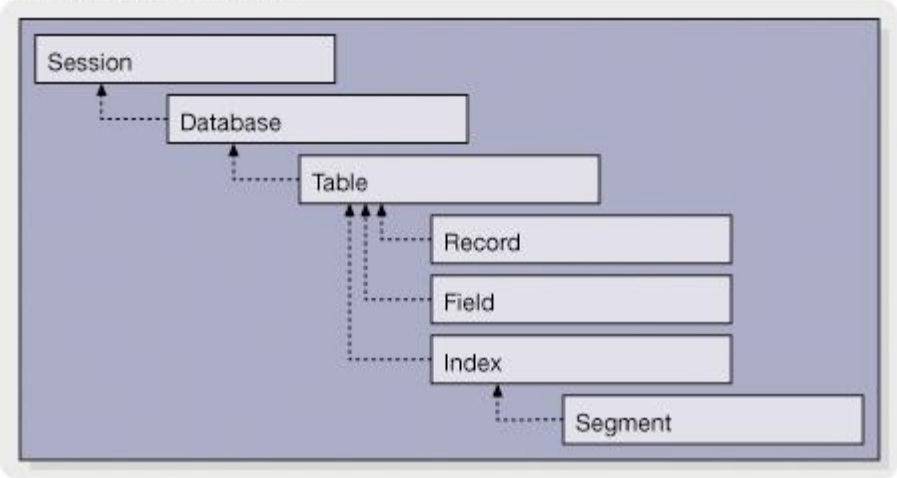
The interfaces are your application's way of accessing the many features and benefits of c-treeACE. The FairCom APIs offer broad support for developers working in a variety of environments. This section lists changes in those APIs.

7.1 FairCom DB API Java - Direct Record Access API for Java

FairCom DB API, short for c-treeACE DataBase, is a high-level, easier-to-use API on top of the c-treeACE ISAM and Low-level APIs. FairCom DB API is intended as the new standard for c-treeACE programming. With this new approach, FairCom has eased development without removing the flexibility and fast performance of our original direct record access support.

FairCom DB API utilizes the simplified concepts of sessions, databases, and tables in addition to the standard concepts of records, fields, indexes, and segments. This database layer allows for effortless and productive management of database systems.

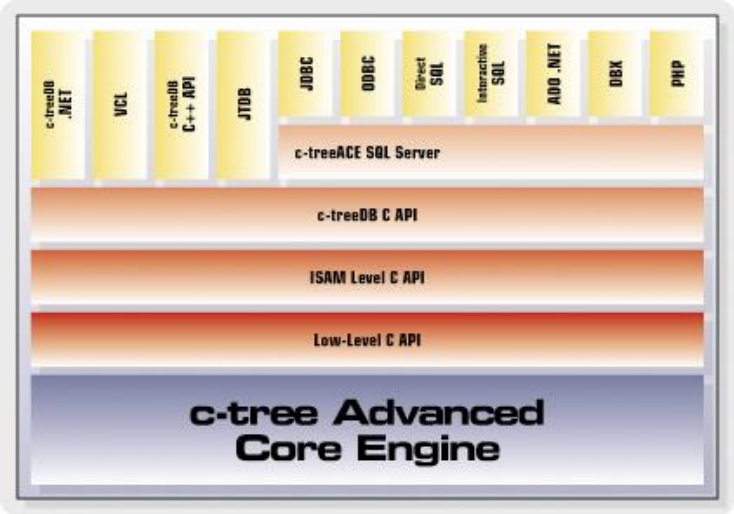
c-treeDB Relationships



Java is a recognized industry standard, cross-platform development environment designed to meet the challenges of application development in the context of heterogeneous, network-wide distributed environments.

JTDB: FairCom DB API Java

FairCom DB API Java, or JTDB, is a native Java interface utilizing the FairCom DB API paradigm. It is a record oriented c-treeACE interface providing advanced, extensive flexibility for the Java developer.



Tutorials and Documentation

A helpful readme is available that will provide a high-level overview to the c-treeACE product and the supported APIs. This file is loaded by default at the end of the installation on Windows platforms, and is included in Unix platforms in *<install directory>/linux*/readme.html*.

To become acquainted with the JTDB, we recommend reviewing the JTDB tutorials located in the following directory of your c-treeACE installation:

```
/<install directory>/<platform>/sdk/ctree.isam.java/tutorials/cmdline
```

To build the tutorials, be sure you have at least JDK 1.6 or newer installed and that it matches the number of bits supported by your c-treeACE installation. For example, if you installed the 64-bit version of c-treeACE, be sure you use the 64-bit version of the JDK available from the command line.

The c-treeACE JTDB documentation is included with this package and can be found in the following locations:

- Linux - *<install directory>/<linux*>/docs/html/sdk/ctree.isam.java*
- Windows - *<install directory>/<win*>\lib\ctree.isam.java\ctreeJTDB-docs.zip*

7.2 ADO.NET

- **c-treeACE SQL ADO.NET Entity Framework Support** - The c-treeACE SQL ADO.NET Data Provider now has support for Entity Framework. When Visual Studio is detected during c-treeACE Professional installation, support is integrated by default.
- **Added support for IDENTITY columns to Entity Framework provider**.

- **Added support for alternative Character Sets** - The previous version had a hard-coded definition for the supported Character Set, which was fixed to iso-8859-1. This character set was good for English and Western European languages. A customer from Hungary complained that some language-specific characters were not properly handled so we decided to add a configurable character set.
- **Clob and Blob types** - The Clob and Blob types are now fully supported.
- **c-treeACE SQL ADO.NET Data Provider Time Fields Now Map to .NET CLR TimeSpan Class** - The c-treeACE SQL Time type (c-tree CT_TIME) was originally mapped to a .NET DateTime type based on the lack of a Time class in the .NET CLR and that a c-treeACE SQL Time value was best represented by a DateTime class due to the point-in-time nature of the pair. It was noted that Microsoft mapped ODBC Time types to TimeSpan in .NET and SQL Server 2008 introduced a Time type that also mapped to this class. c-treeACE SQL Time is now mapped to a .NET TimeSpan type to better align with the Microsoft chosen mapping. This is a compatibility change to be noted in migrating c-treeACE SQL ADO.NET Provider applications to c-treeACE SQL V9.3.
- **TINYINT type missing from EF schema** - We now properly handle the TINIINT type in EF schema.
- **Added GetScale() method to CtreeSQLDatareader** - It is often necessary to get the decimal fields scale value in order to properly format the fields. For this reason we added the new **int GetScale(fieldindex)** method to CtreeSQLDatareader class that returns the field scale.

ADO.NET Entity Framework

The c-treeACE SQL ADO.NET Data Provider has support for Entity Framework through V7.

System Requirements

The minimum development system requirements for c-treeACE SQL ADO.NET Entity Framework support are listed below. Note that they require the complete version (e.g., the complete version, not just the "client" version):

- Visual Studio 2008 Service Pack 1 or greater
- c-treeACE V11.5 requires Microsoft .NET 4.0 Framework.
- c-treeACE V11.0 requires Microsoft .NET 4.0 Framework.
- c-treeACE V10.3 requires Microsoft .NET 4.0 Framework.
- c-treeACE V10 requires at least Framework Version 3.5 SP1

Auto Incrementing Field Type Restriction

Entity Framework Models allow *Int16*, *Int32* or *Int64* field types to be specified as Auto Incrementing in model design. Auto Incrementing fields are also known as Identity fields in some schemas.

c-treeACE SQL allows one user Auto Incrementing field type. Note that c-treeACE already supported a serial segment field, currently used by default as the *ROWID* value. As there is a limitation of one *SRLSEG* field type per data file (table), this precluded the addition of a user-defined field. An IDENTITY attribute is now available for this purpose.

Other Known Limitations

The following are other known c-treeACE SQL limitations that can be encountered when using Entity Framework support. These are in various stages of development. Contact your nearest FairCom office for the latest information concerning specific support for these items.

- The SKIP operator is not currently supported. The SKIP operator is commonly used with the TOP operator for “paging” purposes.
- The EXCEPT operator is not currently supported.
- Parameters are not currently supported in functions and in the TOP operator.
- BIT (Boolean) columns can currently only be tested against 1 or 0 (that is, if (*bitColumn* == 1). Entity Framework requires a test against true/false (for example, if (*bitColumn* == true) or more simply if (*bitColumn*)

7.3 COBOL

Major COBOL support is now available in the new FairCom RTG COBOL product line. Supporting AccuCOBOL, Micro Focus, and Veryant IsCOBOL, FairCom RTG COBOL brings performance, integrity, and advanced SQL access to all of your existing COBOL tables with extreme ease. In many cases, only a single line of COBOL needs to be touched to take advantage of this new freedom. Contact your nearest FairCom office to gain access to this latest technology.

For more information, visit our COBOL page at **www.faircom.com**.

7.4 ISAM

- **ctTransferFile() provides API functions to transfer files** - We have added a convenient API for moving these files to/from a client and server.
- **GETFIL() now has modes to retrieve first and last active partition members** - When working with partitioned files, it is useful to quickly locate the first (the “oldest” when partitioning by date) and last (“newest”) partition members for administrative purposes. Two modes have been added to the **GetCtFileInfo()** function to support this ability.
- **ctGetOpenFiles(), ctGetFileUsers(), and ctGetFileLocks() now also list files that are open pending delete** - We noticed that ctadmn’s option to list all open files did not list transaction-dependent files that had been deleted but whose delete was not yet committed. These files are open but they are marked with a state of 'd', and the function that returns the list of open files only lists the files having a state of 'y' or 'v'. We modified the functions that return a list of open files to also include the files with a state of 'd'.
- **ctLOKDYN(tranPersist) ignores persistent tran lock state** - We now ignore the persistent tran lock state when an unlock request is made outside a transaction. We modified the function that unlocks a data record so that it now rejects an unlock request for a record in a transaction-controlled file when the persistent transaction lock state is enabled only if a transaction is active at the time the unlock request is made. This means that an unlock request that is made outside a transaction will ignore the persistent transaction lock state.

- **ctSETCFG(setcfgDIAGNOSTICS, "LOWL_FILE_IO") supports turning on/off DIAGNOSTICS LOWL_FILE_IO for c-treeACE Server at runtime** - c-treeACE Server now supports turning on or off the DIAGNOSTICS LOWL_FILE_IO configuration option at runtime. This can be done by a call to **ctSETCFG()** or using the **ctadmn** utility.
- **ESTKEY() offers increased precision for large files** - SQL index selection relies heavily on the precision of ESTKEY (used by ESTRNG) to pick the best index. Prior to this revision, ESTKEY returned the estimate to within 1/10 percent. For a file with 100K records, this means that the ESTKEY estimate was +/- 100. For a file with 100M records, ESTKEY was +/- 100K records. This revision increases the ESTKEY precision with the number of keys in the file to attempt to maintain a precision of about 100 records.
- **CTUSER() returns correct data length of custom output data** - An output data length set by the **CTUSER()** function did not limit the amount of data returned to the client. Recently applied **CTUSER()** changes resulted in the output length set by **CTUSER()** to have no effect on the amount of data returned to the client. This is because the communication buffer was set to NULL before calling **CTUSER()**. To resolve this, the internal name of the **CTUSER()** function was changed in *ctsctu_a.c* to **iCTUSER()** and a parameter was added that is a pointer to the output length. **iCTUSER()** returns the output length in this parameter and logic was added to return a proper length of data.

c-treeACE API Functions to Transfer Files

It is often advantageous for applications to deal with files external to the data and index files. Consider the case of a document. While it may be considered as information related to data record, it is useful to maintain the document external to the data file. To allow a convenient method of moving these files between a client and server, a c-treeACE API function is available to transfer the file from client to server or vice versa.

ctTransferFile() is used to initiate this transfer. The function is description is in the **Function Reference** appendix in **c-treeACE Functions to Transfer Files** (page 204).

Configurable Retry Logic for ISAM VLEN_ERR (148) Errors

Under normal operation, a **VLEN_ERR** error (148, **WRTVREC()** cannot fit record at record byte offset) is unexpected. However, this error was encountered when updating a variable-length data record. In addition to diagnostic logging when this error occurs, c-treeACE supports a configuration option to automatically retry an ISAM record add or update that fails with error **VLEN_ERR**. The configuration keyword:

```
VLEN_ERR_RETRY_LIMIT <retries>
```

is used to set the number of times to retry an ISAM add or update operation that fails with error **VLEN_ERR**. If the number of configured retires is exhausted, **VLEN_ERR** is then returned to the calling application. A retry value of 10 to 20 is a recommended starting value, much as the case with **ITIM_ERR** (160) situations.

VLEN_ERR Diagnostic Logging

A record update failed with a **VLEN_ERR** error (148). This error occurs when a call to write data to a variable length record at a specified file offset finds that the size of the region (as indicated by

the total record length in the record header) is too small to hold the record data. When this error occurs the following message is logged to *CTSTATUS.FCS*:

```
WRTVREC: File=<datafilename>, offset=<recordoffset>, [<lockstatus>], trclen=<sizeofcurrentspace>,
varlen=<sizeofnewrecord>, sizhdr=<recordheadersize> header=[<headercontents>] next
header=[<contentsofnextheader>]: 148
```

If this message is observed in the *CTSTATUS.FCS* status log, please collect this file and any other relevant information and contact your nearest FairCom office for further analysis.

7.5 FairCom DB API C and C++

- **ctdbCloneTable() adds new ability to clone a c-treeACE table** - A new function was added to FairCom DB API to clone a table. This function creates a new table with the attributes from an existing source table. This does NOT create a duplicate table -- records are not copied. The **CTTable::Clone()** methods was also added the to FairCom DB API C++, and .NET APIs.
- **Result Set feature** - We have introduced a Result Feature in FairCom DB API. This is a smarter way to set filters based on field criteria. The result set handle is allocated (**ctdbAllocateResultSet**) for a specific table handle, then it is possible to add one or more criteria (**ctdbAddCriteria**). The criteria have a field to be checked against the table handle that owns the result set, one or two values (depending on the comparison operator) and the operator to be used. The operator can be one of: CTIX_EQ, CTIX_NE, CTIX_GT, CTIX_GE, CTIX_LE, CTIX_LT, CTIX_BET, CTIX_BET_IE, CTIX_BET_EI, CTIX_BET_EE or CTIX_NOTBET. When the result set has all the criteria added, it can be turned On/Off (**ctdbResultSetOnOff**) for any record handle that is allocated for the same table handle that owns the result set.
- **ctdbGetCtreeOWNER(VOID)** - Added function to get/set the c-treeACE Owner: In Java, the garbage collector runs in a thread that is not the main application thread and this may cause c-treeACE problems due to the nature of thread handling in c-treeACE and in particular to the “owner” approach c-treeACE implements. Moreover, in FairCom DB API Java there is the necessity to use or reuse a session object (or any “child” object of a session) in a different thread than the one that originally created it. To this end it is good to have a way to get and set the c-treeACE owner in a consistent way across the various supported library models.
- **Added timetostring formats to use 2 digits for hours** - A customer noticed that **ctdbTimeToString()** function uses one or two digits (depending on the value) to represent the hour part, while it always uses two digits for the minutes and the seconds. The FairCom DB API documentation when describing the time formats uses, for instance, HH:MM:SS saying the MM and SS are two digits while says nothing for HH. The customer would like to always use 2 digits also for the hour that would help with data exchange with our database and with reports line-up. We added new formats that enforce two digits also for the hours.
- **Add tables with same filenames to a FairCom DB API database** - A common migration scenario for FairCom DB API users is to import multiple tables into a single database that share the same physical file name, even though they reside in separate directories. This is now supported with the **ctdbAddTableXtd()** API call. This call allows adding a table specifying both a logical and physical table name (the physical table name is the name on disk without the extension and path). The logical table name is the one then normally used for FairCom DB API table operations.

- **Add new table create mode to indicate compressed records** - We added *CTCREATE_COMPRESS* new table create mode to support the new data compression feature.
- **IDENTITY support added to FairCom DB API** - The following functions were added to the FairCom DB API C API for working with Identity fields: **ctdbSetIdentityField()** to Set an Identity Field for a table; **ctdbGetIdentityFieldDetails()** to Retrieve the name, seed and increment of an Identity field; and **ctdbGetLastIdentity()** to Retrieve the last Identity value used for a table.
- **ALTER TABLE Add and Drop Columns supported for Partitioned Files** - The ability to add and drop columns for Partitioned Files via an ALTER TABLE (either via SQL or FairCom DB API) has been added. Previously an invalid argument error was returned (CTDBRET_INVARG) when attempting this operation. For very large data sets this could take time, as currently, every record is visited to update based on the new schema. In addition, if indexes require a rebuild, this will require additional time.
- **New FairCom DB API methods to retrieve partitions** - Additional methods have been added to the c-treeDB.NET API CTTable class to support retrieving the first (oldest) and last (newest) partition members. These methods return the partition rawno value of the partition member, that can then be used to purge or otherwise administer the partition member directly.

Add Tables with Same Filenames to a FairCom DB API Database

A common migration scenario for FairCom DB API users is to import multiple tables into a single database that share the same physical file name, even though they reside in separate directories.

FairCom DB API initially imposed a restriction of a unique physical file per table name in a database. This did not allow, for example, tables with different symbolic names to reference files with the same names but located in separate directories. This ability has been added to c-treeACE V9.3. Tables can now reference files based on a physical file name.

This is now supported with the **ctdbAddTableXtd()** API call.

```
ctdbAddTableXtd(CTHANDLE Handle, pTEXT Name, pTEXT PhysicalName, pTEXT Path)
```

This function allows adding a table to a FairCom DB API database specified with a logical and physical table name (the physical table name is the name on disk without extension and path), the logical table name is the one associated with file operations such as **ctdbOpenTable()**, etc..

When adding tables with a logical name different from the physical name the index names are automatically assigned by FairCom DB API ignoring values stored in the *IFIL* resource.

FairCom DB API Database Dictionary Update

To enable this feature, a new field (index_basename) was added to the database dictionary using space at the beginning of the reserved field. Additional logic to use existing dictionaries without this new field was also included, in which case it is not allowed to have logical names the same as physical names (new error code). Also included in this dictionary change was the removal of *NULFLD* support and *RECBYT* indexes for efficiency.

Importing to SQL

The c-treeACE SQL import utility, **ctsqlimp**, has been updated to allow importing tables with the same physical file name.

New FairCom DB API Ability to Clone a c-treeACE Table

A new function was added to FairCom DB API to “clone” (duplicate) the structure of a table.

```
CTDBRET ctdbCloneTable( CTHANDLE tarTableHandle, pTEXT tarTableName, CTHANDLE srcTableHandle )
```

This function creates a new table with the attributes from an existing source table. **This does NOT create a duplicate table -- records are not copied**.

Notes:
1. This function is only for a *CTSESSION_CTREE* session or for cloning tables from/into different databases.
2. The table to be cloned must be closed as **ctdbCloneTable()** requires an exclusive open on the file.
3. Only the table name is required, not the physical file name.

Example

```
CTHANDLE hTable1;
CTHANDLE hTable2;

ctdbLogon(hSession, "FAIRCOMS", "ADMIN", "ADMIN");

hSession = ctdbAllocSession(CTSESSION_CTREE);
hDatabase = hSession;

hTable1 = ctdbAllocTable(hDatabase);
hTable2 = ctdbAllocTable(hDatabase);

ctdbSetTablePath(hTable1, ".\\data1");
ctdbSetTablePath(hTable2, ".\\data2");

ctdbCloneTable(hTable1, "custmast", hTable2);
```

7.6 FairCom DB API .NET

The following lists new methods added to the FairCom DB API .NET interface (*FairCom.CtreeDb.dll*)

- **ClearSavePoint()** - Added new method
- **CloneTable()** - Added new method
- **GetFirstPartition()** - New method added

- **GetLastPartition()** - New method added
- **LoadCallBackLib()** - Added new method
- **NextInBatch()** - New method added
- **UnloadCallBackLib()** - Added new method

ISAM .NET Additions

A .NET ISAM-based assembly (*FairCom.Isam.dll*) provides direct ISAM functionality to .NET developers for specific needs. Additions have been made to this assembly as follows:

- **CtThrdAttach()** and **CtThrdDetach()** - New methods
- **GetCtFileInfo()** - New modes: partitions; directory; paths; names**GetFieldBinaryFlag()** - New methods
- **GetSymbolicNames()** - New modes: partitions; directory; paths; names
- **SetCndxExpression()** - New methods added

7.7 ODBC

- **ODBC Driver upgraded to be compatible with 64-bit ODBC specifications** - We adjusted to the changes Microsoft made to the 64-bit ODBC driver API function parameter data types.
- **ODBC Driver manager for Unix** - Implemented **SQLGetPrivateProfileString()**, which is a system call on Windows for retrieving ODBC information from the registry.

7.8 Borland VCL/DBX

- **Embarcadero RAD Studio XE and XE2 support** - We have added support for Embarcadero RAD Studio XE and XE2 version, for both VCL and DBX.
- **c-treeACE VCL Method to Compare Records** - To extend FairCom DB API support to VCL the **ctdbCompareRecord()** function was added to the TCtRecord class. This method compares the current CTRecord content against the argument record.
- **VCL - added TCtTable::CloneTable method** - Added the TCtTable::CloneTable method on VCL API. Note that it doesn't apply for DBX because it is not on the SQL level.
- **TCtDataSet Fields Property Type Has Changed in c-treeACE VCL** - The c-treeACE VCL Fields property of the TCtDataSet class had overwritten the generic Fields property of the TDataSet class (the parent of TCtDataSet). c-treeACE VCL used a type of TCtField while the parent field type was of TField. As the TCtField type is meant for internal use, it has been renamed CtField. As a result of this change, c-treeACE VCL now has the default generic Fields property from TDataSet available in TCtDataSet and there are two ways to retrieve field information:
  a) TCtTable.Fields[<field index>] works exactly as TCtTable.FieldByName(<field name>);
  b) With TCtField, specific information remains available as TCtTable.CtFields[<field index>]

c-treeACE VCL Method to Compare Records

To extend FairCom DB API support to VCL the **ctdbCompareRecord()** function was added to the *TCtRecord* class.

```
TCtRecord.CompareRecords( TCtRecord )
```

This method compares the current *CTRecord* content against the argument record.

TctRecord.CompareRecords() returns **CTDBRET_DIFFERENT** (4086) when records do not match, else CTDBRET_OK.

Example

```
// MyTableRec is an extra TCtRecord pointer
void __fastcall TFileIOBase::SaveRecordState(void)
{
        MyTableRec2 = MyTable->ActiveRecord->Duplicate();
}

bool __fastcall TFileIOBase::IsRecordAltered(void)
{
         MyTable->ActiveRecord->Read(); // re-read current record
         if(MyTable->ActiveRecord->CompareRecords(MyTableRec2) == CTDBRET_OK)
                    {return false;}
            else {return true;}
}
```

7.9 PHP

FairCom continues to support the latest PHP versions for easy web access to your data via c-treeACE SQL. This release includes enhancements to 64-bit PHP.

7.10 PYTHON

Python interface to c-treeACE SQL - We implemented a pure Python module called **pyctree** that interfaces with c-treeACE SQL using the DSQL (ctsqlapi) shared library (DLL on Windows) through the well-known ctypes Python module (which allows making calls into a C-based shared library from native Python code). This architecture has been chosen instead of writing an extension in C code because it makes the module immediately available for multiple Python implementations (for example, PyPy and Jython) and does not require any C compiler to install the extension.

7.11 Btrieve

FairCom continues to support a Btrieve interface driver. This driver allows existing Btreive applications to run with c-tree performance and integrity. An easy to use **butil** utility allows easy

data migration into c-treeACE. With this one-time conversion your data is now available from both your existing application AND c-treeACE SQL for the latest access methods.

Contact FairCom to make the easy switch to c-treeACE from your existing Btrieve application.

8. c-treeACE SQL Advances

The c-treeACE SQL engine allows applications to access your data using this industry-standard query language. The great strides we have made in optimizing our SQL subsystem are described in this section.

8.1 Conditional ISQL Expressions

The ISQL utility is an easy interface to work directly with the c-treeACE SQL database engine. The ability to conditionally work with SQL statements based on an existing database state provides a flexible method of updating and administratively maintaining tables and database information. The c-treeACE SQL ISQL utility now supports conditional expressions to make this possible.

Syntax

```
{IF [NOT] EXISTS} | {IF_EXISTS} | {IF_NOT_EXISTS} (<query>)
    {BEGIN
            <list of statements>

    END} | { <statement> }

[ELSE
    {BEGIN
            <list of statements>
    END} | { <statement> } ]
```

Conditional expressions can be nested. The following example checks for the existence of a table and creates it if it does not exist. The nested condition updates an existing table if it is required.

Example

```
IF NOT EXISTS (SELECT * FROM systables where tbl = 'custmast')
BEGIN
    CREATE TABLE custmast (name CHAR(10), custid INTEGER IDENTITY (1,1), balance MONEY, modtime TIMESTAMP);
END
ELSE
BEGIN
    IF NOT EXISTS (SELECT * FROM syscolumns where tbl='custmast' AND col='modtime')
            BEGIN
                    ALTER TABLE custmast ADD (modtime TIMESTAMP);
            END
    ELSE
            BEGIN
                    SELECT * FROM custmast;
            END
END
COMMIT WORK;
```

The following keywords have been added to the ISQL lexicon to support conditional expressions:

- IF
- IF_EXISTS
- IF_NOT_EXISTS
- BEGIN
- ELSE
- END

Statements that use these keywords as identifiers should enclose them in double quotes.

8.2 Set the Connection Node Name in c-treeACE SQL

c-treeACE has an ability to identify client connections by node name, which is a 32-character ASCII name that can be assigned by the client application. To provide an easy ability to do this in SQL, a built-in stored procedure is now available to set this by a connection:

```
fc_set_nodename( VARCHAR nodename )
```

No error or result sets are returned by **fc_set_nodename()**. It can be called multiple times within a SQL session to change the node name. When called with an empty string it clears the node name.

8.3 Duplicate Index Names Now Allowed in c-treeACE SQL Databases

c-treeACE SQL imposed a restriction of only one instance of an index name per database. For example, an inventory database with a customer table and an invoice table could not have an index on each table named “date.” This was loosely implied by the fact that the database engine did not create index files with a unique name per table. This enhancement removes this restriction by pre-pending the table name to the index file name, thus removing the index file name collision problem.

This support could cause unexpected problems in applications that utilize a FairCom DB API API and have hard-coded index names for tables created using c-treeACE SQL. Therefore, a configuration keyword has been added to revert to the original behavior.

```
SQL_OPTION NO_DUP_IDXNAME
```

With this keyword present in the *ctsrvr.cfg* configuration file, index naming is as before and it is not possible to create indexes with the same name even if belonging to different tables.

Impact on Index Names for Constraints

Index names generated by c-treeACE SQL for constraints (primary key, foreign key, checks) previously contained a unique table ID. This ID is now redundant as a table name will be prepended to the index name with the addition of support for duplicate index names in a database.

8.4 c-treeACE SQL Option for Certain Slow Parameterized Queries

A set of parameterized SQL queries, with some literals, was found to run slowly. When optimizing a parameterized query, the c-treeACE SQL optimizer uses an average selectivity. If the specified literal value has a selectivity of a great enough magnitude away from the average, the generated plan may be less than optimal. A new configuration option allows a controlled selectivity cutoff:

```
SETENV DH_REBUILD_SEL_CUTOFF
```

This controls the regeneration of the plan for queries with parameter references, during execution, based on the value set: -1 < n <= 100.

- If the value is set to -1, query plans are never regenerated during execution.
- If the value is set to 0, query plans are always regenerated during execution.
- If the value is set between 1 and 100, query plans will be regenerated whenever the percentage difference in selectivity (i.e., the selectivity that was used during the generation of the original plan and the selectivity for the actual parameter values passed during execution) exceeds this value. For example, if the value is set to 25, the plan will be regenerated only if the selectivity of the actual parameter is different from the original selectivity than 25%.

Example

```
SETENV DH_REBUILD_SEL_CUTOFF=25
Default: (-1) Query plans are never regenerated during execution.
```

8.5 c-treeACE SQL Open Table Configuration

A server configuration keyword is available to set the number of tables to keep open per c-treeACE SQL connection, although the table is not in current use. This improves performance in some cases by avoiding the I/O and caching overhead of opening and closing files.

```
SQL_KEEP_OPEN_TABLES <number of tables>
```

The default is 60. Specifying a negative value will enable the default.

8.6 Support for Imported Hidden Fields with c-treeACE SQL

c-treeACE has long supported the ability to “hide” fields from casual schema inspection using a “_” prefix to the column name. These fields were then skipped when importing into c-treeACE SQL. However, this could lead to problems when accessing schema information via interfaces other than c-treeACE SQL. For example, a c-treeACE SQL Server linked to a Microsoft SQL Server via ODBC would not properly work with these tables, as the enumeration (ORDINAL_POSITION) of the columns did not accurately reflect actual available columns.

An attribute has been added to the c-treeACE SQL system table syscolumns (*syscolumns.logicalid*) that logically identifies columns when a logical ordering is required. The field is NULL by default.

Existing tables with hidden fields should be re-imported if they exhibit problems. The *syscolumns.logicalid* can also be manually updated following an automatic database upgrade with the new system table format.

8.7 Faster c-treeACE SQL Database Conversions

As new features and abilities are added to c-treeACE SQL, it is often necessary to update the core SQL system tables with new or modified attributes. When upgrading c-treeACE SQL versions, c-treeACE attempts to detect previous database versions and automatically convert them at startup for seamless migrations.

Recent conversions demonstrated exceptionally long conversion times when converting multiple databases. Internally, c-treeACE SQL completes this process under full transaction control. However, the transactions were not always handled most efficiently. By adjusting transaction logic, conversion times have been substantially improved.

It was observed that the conversion could fail without a logged message. Before conversion, a backup timestamped copy of the database file is made (*dbname.fdk*). Errors that occur during conversion are now logged to *CTSTATUS.FCS.* It is recommended to examine the *CTSTATUS.FCS* status log after a server upgrade and ensure a complete and successful conversion took place.

Note: To disable database conversion at server startup, the c-treeACE SQL configuration keyword `SQL_OPTION NO_DB_CONVERSION` can be used.

9. GUI Tools

Numerous GUI tools are included with c-treeACE. This section begins with a refresher course on those tools, including the new Java-based GUI tools that are now available from FairCom. The latest updates to these utilities are summarized in: Recent Changes to the GUI Utilities (page 107)

9.1 GUI Refresher Course

A set of GUI utilities is included with the FairCom product. To help you see the selection of utilities at your fingertips, they are listed in this section. For those of you who are new to FairCom, this section is a starting place for unlocking the power of this environment. If you have been using FairCom for a while, think of this section as a refresher course.

Data Management

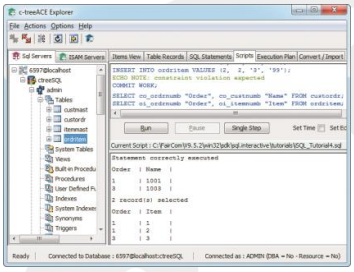
c-treeACE Explorer - NEW for V10!

New for V10: This GUI tool is written in cross-platform Java for ISAM and SQL Administration.

Area: ISAM & SQL

Platform: All Platforms

Purpose: Administration, Monitoring



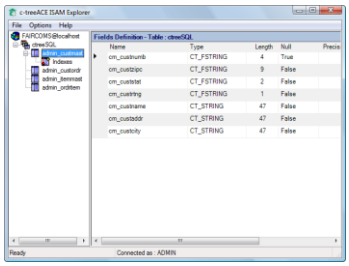
c-treeACE ISAM Explorer

ISAM Administration tool to create, view, and modify databases.

Area: ISAM

Platform: Windows

Purpose: Administration



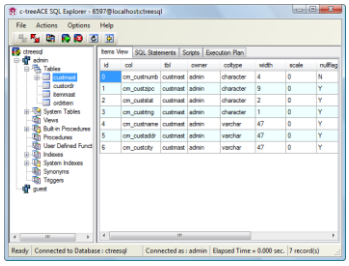
c-treeACE SQL Explorer

SQL administration tool to work with tables, indexes, views, synonyms, procedures, user-defined functions and triggers; execute single SQL statements and SQL scripts; and view execution plans.

Area: SQL

Platform: Windows

Purpose: Administration



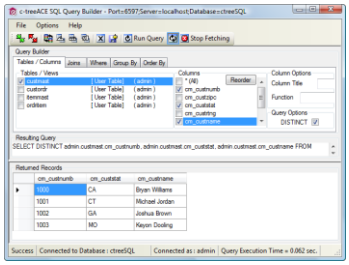
c-treeACE Query Builder

Tool for creating simple SQL queries to send to the c-treeACE Server.

Area: SQL

Platform: Windows

Purpose: Administration



Monitoring

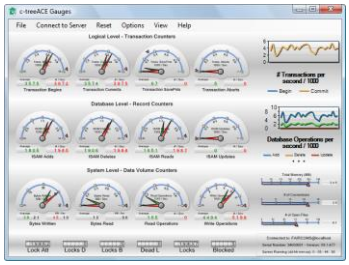
c-treeACE Gauges

Performance monitoring tool to display metrics for analyzing server performance.

Area: System

Platform: Windows

Purpose: Monitoring



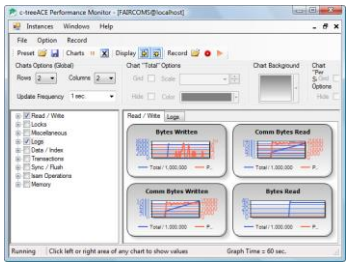
c-treeACE Performance Monitor

Performance monitoring tool providing graphs showing data throughput activity.

Area: System

Platform: Windows

Purpose: Administration, Monitoring



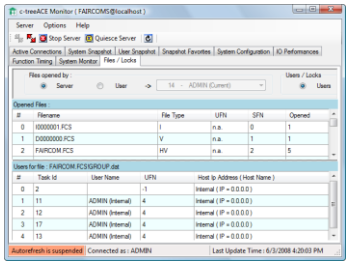
c-treeACE Monitor

Tool to monitor and report server activity: events, configuration settings, connected users, etc.

Area: System

Platform: .NET version: Windows only; Java version: All Platforms

Purpose: Administration, Monitoring



System Admin

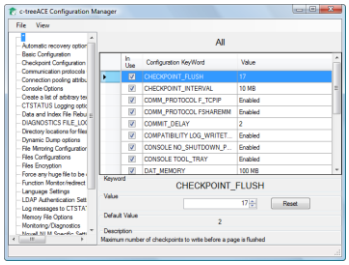
c-treeACE Configuration Manager

Administration tool to manage and keep track of configuration keywords.

Area: System

Platform: Windows

Purpose: Administration



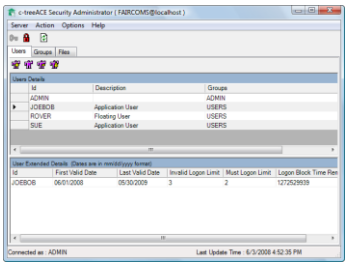
c-treeACE Security Administrator

Security Administration tool to manage users and user groups.

Area: System

Platform: Windows

Purpose: Administration



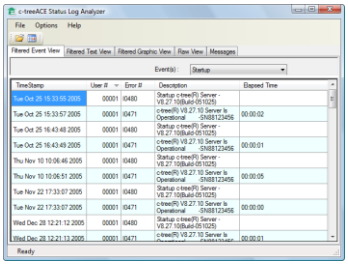
c-treeACE Status Log Analyzer

Maintenance tool to analyze the Status Log File (*CTSTATUS.FCS*), which contains important information regarding the status of the server.

Area: System

Platform: Windows

Purpose: Administration, Maintenance, Monitoring



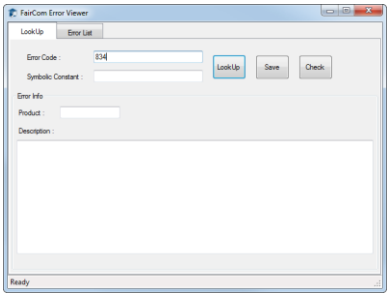
FairCom Error Viewer

Administration tool to look up the meaning of c-treeACE error codes.

Area: System

Platform: Windows

Purpose: Administration, Developer Support



Performance Benchmarking

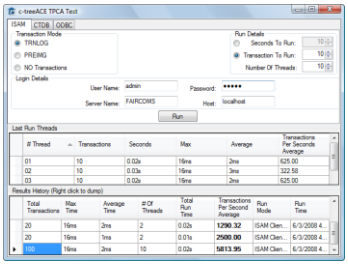
c-treeACE TPCA Test

GUI Version of **cttpca** test program, a benchmark utility that implements Transaction Processing Performance Council Benchmark A test (TPC-A) to emulate an update intensive database environment. (Provided as a sample.)

Area: System

Platform: Windows

Purpose: Performance



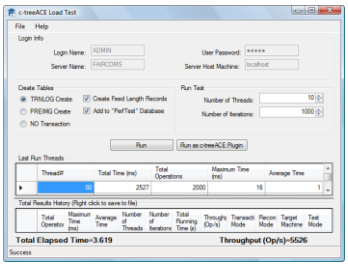
c-treeACE Load Test

GUI Version of **cttctx** test program, a benchmark utility that measures the performance of record inserts, reads, and deletes in a given set of data files. (Provided as a sample.)

Area: System

Platform: Windows

Purpose: Performance



Advanced Developer Tools

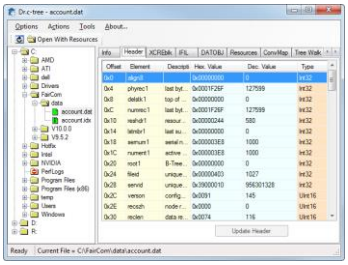
Dr. c-tree

Low-level developer tool to work with data files, index files, and data dictionaries.

Area: Low-level ISAM

Platform: Java version: All Platforms; .NET version: Windows only

Purpose: Developer Support



9.2 Recent Changes to the GUI Utilities

This section summarizes recent changes to the FairCom GUI utilities. Those of you who already use these utilities will want to review this section to stay you up-to-date with the latest corrections and enhancements.

New Java-based GUI Utilities Now Available

The c-treeACE GUI tools have been rewritten in Java to provide the same advanced graphical interface to all platforms supporting the Java Runtime Environment (JRE). The Java tools have the same look and functionality as the c-treeACE .NET-based tools. And, being authored with Java, they are now cross platform, supporting Windows, Mac, Solaris, AIX, Linux and other Unix environments.

The c-treeACE Explorer combines c-treeACE ISAM Explorer and c-treeACE SQL Explorer views in a single, cross-platform tool.

These Java-based tools include several new features originally not available in the .NET-based tools:

- ISAM search by key value
- Open external tables
- XML data output

c-treeACE SQL Explorer Modify Table

Corrections have been made to address two issues with Modify Table in c-treeACE SQL Explorer:

1. When creating a TRNLOG table with conditional indices set at creation time, none were created as conditional. **CndxCompile** failed with **NOTACTIVE**. This has been corrected by changing the logic to reopen the table before it calls additional methods.
2. When an index was created and the filename was left as “(Default)”, it created a file on disk named *(Default).idx*. This behavior has been changed to set the filename to the logical name.

c-treeACE ISAM Explorer Conditional Indices

Issues were discovered with **ctdbUpdateCndxIndex** which affected the ability to add conditional indices in c-treeACE ISAM Explorer. These issues have been corrected so that conditional indices can be added using c-treeACE ISAM Explorer.

ctdbSetIndexCndxExpr() was changed to allow it to work with existing indices. When used with existing indices, **ctdbAlterTable()** should use *CTDB_ALTER_INDEX* mode to ensure the conditional index changes are applied.

This change also includes a small fix for non-SQL tables in a SQL database, which replaces a **CTDBRET_INTERNAL** error with the underlying **isam_err**.

c-treeACE ISAM Explorer GetCndxIndex Argument

It was found that c-treeACE ISAM Explorer had an incorrect argument for **GetCndxIndex** in several places. This function has been corrected so that it now performs correctly.

10. Command-Line Utilities

This section covers the comprehensive set of command-line utilities included with c-treeACE V10. Considerable enhancements have been made to a number of these utilities. You will find those changes in:
Recent Changes to the Command-Line Utilities (page 120).

To understand the vast power of the FairCom product, this section begins with a high-level overview of the command-line utilities. For a complete understanding of these utilities, see the FairCom documentation.

10.1 Command-Line Utilities Refresher Course

To help you understand the comprehensive set of utilities provided with the FairCom product, this section provides listing of these utilities. If you are new to FairCom, this is a starting place for unlocking the power of this environment. If you have been using FairCom for a while, this section provides a refresher course.

In this section, the command-line utilities are in alphabetical order. The intended purpose (administration, data integrity, etc.) is shown under **Purpose**. When a GUI version is available, the name of that tool is indicated in **GUI Version**.

Many of these utilities have recently been enhanced or fixed; the details of those updates are listed in the next sections.

ctadmn - c-treeACE Server Administrator Utility

Manage users, groups, and files; monitor users logged on to the server; and disconnect a user from the server.

Purpose: Administration, Security

GUI Version: c-treeACE Security Administrator; c-treeACE Monitor

ctclntrn - Clean Transaction Mark Utility

“Cleans” the high-water transactions marks within a c-treeACE index.

Purpose: Maintenance

ctcmdset - Create Encrypted Password File

Create an encrypted password file from a text file.

Purpose: Administration, Security

ctcmpcif - IFIL-based Compact Utility

Reads the IFIL structure from the data file and calls **CompactIFileXtd()** and **RebuildIFileXtd()** to compact and rebuild it and its associated indexes.

Purpose: Administration, Maintenance

ctcv43 - Convert V4.3 Data File to c-tree Plus Format

Convert an older c-tree V4.3 data file to the current c-treeACE format.

Purpose: Administration, Conversion, Migration, Version Upgrade

ctcv67 - Extended File Conversion Utility

Convert data and index files from Standard to Extended c-treeACE format; adding huge file support, changing segment support, and compacting the output file

Purpose: Administration, Conversion, Migration, Version Upgrade

ctdidx - Create Flat Key File

Create a flat key file containing each key value at full, uncompressed length followed by the associated long integer, 4-byte, record position.

Purpose: Administration, Conversion, Migration, Version Upgrade

ctdmpidx - Index Dump Utility

View index header details, nodes, and key values.

Purpose: Diagnostics

ctdump - Dynamic Dump Utility

Back up data files controlled by c-treeACE Server.

Purpose: Administration, Recovery

ctencrypt - Utility to Change Master Password

New! Change the master password for specified c-treeACE data, index, and transaction log files.

Purpose: Administration, Security

ctfchk - File Checksum Utility

Calculate the checksum on all active records in a fixed-length data file, which can be used to compare the record contents of two fixed-length data files.

Purpose: Diagnostics

ctfdmp - Roll Forward Dump System Recovery Utility

Restore data to a given time following a **ctrdmp** restore.

Purpose: Administration, Recovery

ctfileid - Update File IDs

Update the file IDs of the file header so different applications and/or client nodes can refer to the same file with different names (e.g., to accommodate different drive or path mappings).

Purpose: Administration, Data Integrity, Maintenance

ctflat - Create c-treeACE Plus File from a Flat File

Create a fixed-length data file with a specified record length from an existing flat file.

Purpose: Administration, Migration

ctflvrfy - Index Verify Utility

Calls the **ctVERIFY()** and **chkidx()** functions to verify and optionally inspect an index file.

Purpose: Diagnostics

cthghtrn - Display High-Water Mark for Transactions

Shows the high-water transaction marks within an index file (used when the transaction mark number gets too large).

Purpose: Diagnostics

ctidmp - Examine Dump Files Utility

List all or part of a dynamic dump file.

Purpose: Diagnostics

ctin43 - Create Flat Key File

Create a flat key file from a c-tree V4.3 index file by dumping keys for designated index member to a flat file (index must first be converted from V4.1F-V4.3C using **ctin43** and then **ctindx**).

Purpose: Administration, Conversion, Migration, Version Upgrade

ctindx - Create Index File from Flat Key File

Create an index file from a flat key file, such as the flat key file produced by **ctin43**.

Purpose: Administration, Conversion, Migration, Version Upgrade

ctinfo - Incremental ISAM Utility

Retrieves *IFIL* and *DODA* structures and *XCREblk* extended header information (if available) from a c-treeACE file.

Purpose: Diagnostics

ctldmp - Transaction Log Dump

Perform a partial dump of the transaction logs to assist the developer in problem resolution.

Purpose: Diagnostics

ctmark - Simulated Data Generation Test

Use a parameter file that contains 128-byte fixed-length records keyed by unique number, name, and department code (the names vary in length, with a typical amount of leading character redundancy; the department codes are highly repetitive).

Purpose: Sample Programs

ctmove - Move Files without Stopping c-treeACE Server

Move files without stopping c-treeACE Server to minimize downtime during file recovery.

Purpose: Administration, Maintenance, Recovery

ctmtap - Low-Level Multi-Threaded Test

A multi-threaded version of **ctmark** that demonstrates FairCom's Multi-thread API by running a trial of database operations.

Purpose: Sample Programs

ctmtlk - Lock Operation Test Utility

Ensure proper locking with the multi-threaded stand-alone model.

Not built by default; source available at *..sdk\Xtras\ctree.samples\isam*

Purpose: Diagnostics

ctquiet - Quiesce c-treeACE Utility

Temporarily suspend operation of the server for specific files/actions.

Purpose: Administration, Maintenance

ctrbldif - IFIL-based Rebuild Utility

Reads the IFIL structure from the header and calls **RebuildIFileXtd()** to rebuild the file and its indexes.

Purpose: Administration, Data Integrity, Maintenance

ctrdmp - Rollback Dump Recovery Utility

Restore dumps created with **ctdump**; also performs data roll backs.

Purpose: Administration, Recovery

ctredirect - IFIL Update Utility for Redirected Filenames

Allows a file to be repositioned into another location following autorecovery, dynamic dump restore, or replication.

Purpose: Administration, Maintenance, Recovery

ctsbld - Superfile Rebuild Pre-Pass

Verifies the deleted space in the data file and recreates, in place, any companion index files.

Purpose: Administration, Maintenance, Superfile Integrity

ctsbldm - Rebuild Superfile Index Members

Rebuild superfile index members without having to process the entire superfile (assuming the data file is not corrupt, contains an *IFIL* describing all the indexes, and the superfile is intact except for the indexes in question).

Purpose: Administration, Maintenance, Superfile Integrity

ctscmp - Superfile Compact Utility

Compact and rebuild a superfile and all of its members (provided the IFIL resources are present in the superfile data members).

Purpose: Administration, Maintenance, Superfile Integrity

ctsfex - Superfile Data Export Utility

Export data from a superfile to another superfile or to individual files identical to the superfile members.

Purpose: Administration, Maintenance, Superfile Integrity

ctsfil - Superfile Contents Utility

List members of the superfile, indicating if each member is an index file, fixed-length data file, or variable-length data file.

Purpose: Administration, Maintenance, Superfile Integrity

ctsqlbigint - c-treeACE SQL Big Integer Fix Utility

Detect and correct BIGINT issues on HIGH LOW machines should they occur.

Area: SQL

Purpose: Diagnostics, Administration, Data Integrity, SQL Tools

ctsqlcdb - c-treeACE SQL Database Maintenance Utility

Create, add, or drop a c-treeACE SQL database from the server.

Area: SQL

Purpose: Administration, Maintenance, SQL Tools

ctsqlimp - c-treeACE SQL Import Utility

“Register” (link) existing files to a c-treeACE SQL database without modifying the table structure so the files remain accessible from the original application.

Area: SQL

Purpose: Administration, Conversion, Migration, SQL Tools

ctsqlmdd - c-treeACE SQL Move Database Utility

Move a database from one system to another, including path changes to be reflected in the session and database dictionaries.

Area: SQL

Purpose: Administration, SQL Tools

ctsqlutl - c-treeACE SQL Table Maintenance Utility

Perform maintenance on databases, including renaming tables and columns.

Area: SQL

Purpose: Administration, Maintenance, SQL Tools

ctstat - Statistics Utility

Display statistics collected by the server for real-time monitoring of critical operations.

Purpose: Monitoring, Diagnostics

GUI: c-treeACE Monitor, c-treeACE Gauges

ctstress - Perform Record Operations

Perform various database operations for use as a test.

Not built by default; source at *..\Xtras\ctree.samples\special\utils*

Purpose: Sample Programs

ctsysm - Server Status Log Monitoring Utility

Monitor c-treeACE Server status log entries.

Purpose: Administration, Monitoring

cttctx - Performance Test Utility

Simulate high load conditions as a client benchmark test of operations for profiling performance and verifying application performance.

GUI: c-treeLoadTest

Purpose: Performance, Sample Programs

cttpca - TPC-A Benchmark Test Program

A benchmark utility that implements Transaction Processing Performance Council Benchmark A test (TPC-A) to emulate an update-intensive database environment.

GUI: c-treeACE TPCA Test

Purpose: Sample Programs

cttrap - Communications Trap Playback Utility

A multi-threaded client application that “plays back” a TRAP_COMM log file.

Purpose: Diagnostics

cttrnmod - Change Transaction Mode Utility

Display or change the transaction status of a data file and its associated index files.

Purpose: Diagnostics

ctunf1 - File Reformatting Utility

Reformat a file in place provided the new alignment restriction does not cause the data record contents to become misaligned.

Purpose: Administration, Conversion

ctunf2 - UNIFRMAT File Reformatting Utility

Convert UNIFRMAT files to HIGH_LOW.

Purpose: Administration, Conversion, Migration

ctupdpad - Update Pad Resource Utility

Change the padding and delimiter characters in the padding resource.

Purpose: Administration, Maintenance

ctvfyidx - Index Verify Utility

Verify the integrity of an index.

Purpose: Diagnostics

ctvlqa - Variable-length Quality Assurance Utility

A torture and performance test that generates random data to build generic records based on the file structure layout from **ctixmg**.

Purpose: Data Integrity, Sample Programs

dbdump - Data Unload Utility

Write data from a c-treeACE SQL file to an output file.

Area: SQL

Purpose: Administration, Recovery

dbload - Data Load Utility

Load records from an input data file into database tables.

Area: SQL

Purpose: Administration, Recovery, SQL Tools

dbschema - Schema Export Utility

Recreate specified database elements and data.

Area: SQL

Purpose: Administration, Recovery, SQL Tools

fkverify - SQL Foreign Key Constraints

Check an existing database for the foreign key constraint issues.

Area: SQL

Purpose: Diagnostics, SQL Tools

ISQL - Interactive SQL Utility

A standard command-processing interface to the SQL database engine.

Area: SQL

Purpose: Administration, SQL Tools

GUI: c-treeACE Query Builder; c-treeACE SQL Explorer

mtmake - Build Library Utility

Generate a customized makefile for a specific environment, based on selected menu options, to build libraries for ISAM, low-level, and FairCom DB API APIs.

Purpose: Developer Support

sa_admin - User and Group Administration Tool

A command-line version of the system administrator program, which can be used in script files to ease initial setup.

Purpose: Administration

GUI: c-treeACE Security Administrator

10.2 Recent Changes to the Command-Line Utilities

This section lists the enhancements and other changes to the FairCom command-line utilities. If you are already familiar with these utilities, be sure to read this section to stay up-to-date with the latest improvements.

ctadmn c-treeACE Server Administration Utility

When reporting the server memory usage, **ctadmn** now considers the memory dedicated to the cache in the calculation.

For SQL connections, the client activity monitoring option now reports the elapsed time of each request and whether the server is processing a request for that particular client, as well as the function being executed.

ctcmdset Create Encrypted Password File

The **ctcmdset** utility is used to create an encrypted password file from a text file. An issue was discovered when enabling sequence number support in standalone mode. This issue has now been corrected.

ctcmpcif IFIL-based Compact Utility

Two new command-line options were added to this tool:

- *-redosrl* Reassign serial numbers instead of copying the values from the original file.
- *-x8* Use extended create blocks read from data and index files, which creates new compact index files using the same extended create block settings as the original file. Note that, even without this option, some of the extended create block settings are always applied to the new files (e.g., huge file, extended header, and 6-byte transaction number).

In addition to the new parameters listed above, the utility was enhanced so that segmented files can be compacted.

ctcv67 Extended File Conversion Utility

New command-line options were added to encrypt the file during the conversion, set the large file extension size, and set the temporary directory. In addition, a correction was made to address an issue in which the tool would return “succeed” in certain circumstances when the file was not fully converted.

ctencrypt Utility to Change Master Password

The new **ctencrypt** utility is used to change the master password. c-treeACE’s advanced encryption feature uses a master password to encrypt the file-specific advanced encryption key in c-treeACE data, index, and transaction log files. This tool is a standalone utility that can be used to change the master password for specified c-treeACE data, index, and transaction log files.

ctfchk Checksum Utility

The new **ctfchk** Checksum utility is a standalone c-treeACE utility that reads active records from a fixed-length c-treeACE data file in physical order and outputs a checksum and active record count. It uses the CRC-32 algorithm to compute the checksum. This utility can be used to compare the record contents of two fixed length data files. However, two c-treeACE data files that contain the same active record images in the same order will generate identical checksums. Two c-treeACE data files whose active record contents differ will generate different checksums (subject to the limitations of the CRC-32 algorithm).

ctidmp Examine Dump Files Utility

The **ctidmp** Examine Dump Files utility was not displaying the contents for files over 4GB. This has been corrected.

ctinfo Incremental ISAM Utility

The **ctinfo** utility has been updated to display the correct superfile member number. This utility also displays the extended file mode attributes, applicable to partitioned files.

ctitop Utility Creates OTP and Parameter File

A new tool, **ctitop**, was added to convert a file with embedded *IFIL* and *DODA* resources to an OTP file and related parameter file.

ctixmg Updated to Support ctCAMO Encryption

The **ctixmg** example program now supports ctCAMO by default. CAMO or "Camouflage" is an older, legacy method of hiding data, which is not a standards-conforming encryption scheme, such as AES. It is not intended as a replacement for Advanced Encryption or other security systems. The advanced encryption, AES32, is supported by use of an additional keyword in the server configuration file (*ctsrvr.cfg*): `ADVANCED_ENCRYPTION`.

ctmtap Multi-threaded API Sample

Improvements were made to the **ctmtap** test program, including a new command-line option. The new option, *-X*, times calls made to the **CTUSERX()** function. For example, the command-line option *-X(i,o)* causes the program to run the **CTUSERX()** test with input buffer size *i* and output buffer size *o* (in bytes).

ctotoi Utility Adds IFIL and DODA Resources

The new **ctotoi** utility adds IFIL and DODA resources to a c-treeACE data file based on an OTP file.

ctpathmigr Utility to Change Database Path Separators

The new **ctpathmigr** utility changes database and IFIL resource path separators when migrating databases between Unix and Windows platforms.

ctquiet Quiesce c-treeACE Utility

The new **ctquiet** utility was added for quiescing the server through a script file.

ctrbldif IFIL-based Rebuild Utility

The following changes were applied to the **ctrbldif** utility:

Added two new command line options:

- *-sortmem=<N>* Sets sort memory to N KB in standalone builds. Using this option can increase the rebuild speed for large files.
- *-x8* Use extended create blocks read from data and index files, which is useful for creating rebuilt index files using the same extended create block settings as the original file. Note that, even without this option, some of the extended create block settings are always applied to the new files (e.g., huge file, extended header, and 6-byte transaction number).

Changed default sector size from 16 (page size 2048) to 64 (page size 8192) which is the default setting for c-treeACE Server.

Changed *temppath* from NULL (which uses the system temp file path) to the current directory.

Corrected a situation which, under certain circumstances, could cause the utility to return successful and not rebuild one index.

ctredirect Utility to Change IFIL Filename Information

The new **ctredirect** utility updates *IFIL* filenames for redirection to alternate locations. The Redirect feature allows a file originating in one directory structure to be repositioned into another directory location following automatic recovery, dynamic dump restore, or replication. When copying c-treeACE files from one directory location to another (on the same system or on a different system) and accessing them in their new location, it is necessary to update any filename paths in a c-treeACE data file's *IFIL* resource.

ctscmp Superfile Compact Utility

The superfile compact utility, **ctscmp**, now creates indexes with the same distinct key count attribute as the original indexes. In addition, this utility has been updated to correctly open files when the resources and the *ikeydup* attribute in the IFIL resource do not match the *keydup* field in the index file's header.

ctsqlimp Utility

The **ctsqlimp** utility, which allows you to import existing c-tree Plus files into a c-treeACE SQL database, now includes a command line option (*-q <prefix>*) so you can append a prefix to the name of the table being imported.

ctsqlimp has also been updated to take advantage of the ability to import multiple files with the same physical name into an SQL database. Simply use the *-n* (symbolic) name option on import to uniquely identify each table in c-treeACE SQL

ctstat Statistics Utility

A new command-line option (*-mu*) has been added to the **cstat** utility. This option allows you to unload the module debug symbols using ctstat.

ctstress Perform Record Operations on Files

This sample program performs various database operations for use as a test program. The source has been moved to the *..\sdk\Xtras\ctree.samples\special\utils* folder.

cttctx Performance Test Utility

The **cttctx** utility now supports timing options on x86/x64 Solaris systems.

A new command line option was added to monitor only the elapsed time, and not include the individual call times.

Fixed a problem with the directory creation when running in a heterogeneous environment (Windows/Unix).

Fixed a problem with the command-line option that determines which operations (add, read, delete) it performs. In some cases, the result presented was 0 due to an internal reading error.

cttrap Communications Trap Playback Utility

A correction addressed an issue in which the utility was hanging when it used the Shared Memory Protocol and an error **LMTC_ERR** (530) happened.

A new option was added for faster reply: *-x* skip serialization.

cttrnmod Change Transaction Mode Utility

This version of **cttrnmod** includes an important improvement: If one data file without its indexes is required to change the transaction mode, the utility now does the change, and when the file is rebuilt later, the indexes will reflect the transaction mode change.

A new command line option was added: *-n <sect>* This option allows you to specify the sector size (*sect*) in the standalone mode.

ctunf1 File Format Conversion Utility

It was found that the **ctunf1** file conversion utility failed to reverse the bytes of some of the fields in the extended header of a c-treeACE file. The definitions of recently added fields were not added to the conversion map structures used by the **ctunf1** utility. These missing fields have been added to ensure a complete conversion.

ctupdpad Pad Resource Utility

The **ctupdpad** utility, which allows you to change the padding and delimiter characters in the FairCom DB API padding resource, has a new option (*-n*) to set the sect size.

fkverify SQL Foreign Key Constraints Utility

Improved performance in the process by including a WHERE clause to remove NULL keys from consideration.

Utilities Updated for Use with Advanced Encryption

The **ctrdmp** restore utility and the **ctfdmp** (forward roll) utility now detect when transaction logs are using advanced encryption and when data or index files that are referenced in the transaction logs use advanced encryption. When the transaction logs use advanced encryption, these utilities prompt for the master password after opening the first transaction log, which is essentially at the beginning of the restore or forward roll process. When data or index files referenced in the logs use advanced encryption, the user is prompted for the master password the first time it finds a reference to a file that uses advanced encryption. This requires the *ctsrvr.pvf* password file to be created and present.

The **ctdmpidx** dump index utility and **ctvfyidx** index verify utility now also detect when the index file that is being inspected is encrypted with advanced encryption. In this situation, the utilities turn on c-treeACE's advanced encryption support and prompt for the master encryption password just as the server would at startup.

11. New Performance Tuning Options

New options that allow administrators to fine tune system performance are listed in this section.

11.1 COMMIT_DELAY Value Modified for Windows Systems

Our experience with Windows shows that using `COMMIT_DELAY` can dramatically improve performance of *TRNLOG* transactions. This default is now `COMMIT_DELAY 2` for Windows systems.

11.2 Reduce Performance Impact of Dynamic Dump

When a dynamic dump runs, its disk read and write operations can slow the performance of database operations. The c-treeACE Server now supports an option that allows an administrator to reduce the performance impact of a dynamic dump. The new `DYNAMIC_DUMP_DEFER <milliseconds>` option sets a time in milliseconds that the dynamic dump thread will sleep after each write of a 64KB block of data to the dump backup file.

11.3 Transaction Log Index ON by Default

ctLOGIDX is an index file mode permitting faster index automatic recovery during c-treeACE startup. Transaction controlled indexes with this file mode are recovered more quickly than with the standard transaction processing file mode *ctTRNLOG* alone. This feature can greatly reduce recovery times for large indexes.

The c-treeACE configuration option `FORCE_LOGIDX` now defaults to `YES` instead of `NO`, meaning *ctLOGIDX* is forced on for all *ctTRNLOG* index files. The `FORCE_LOGIDX` configuration option can be used to change the default if desired.

11.4 RECOVER_MEMLOG for Faster Automatic Recovery with Advanced Encryption

When transaction logs are encrypted with advanced encryption enabled, FairCom recommends the use of the `RECOVER_MEMLOG` configuration option to load the transaction logs into memory before recovery. This provides a faster decryption as data is already decrypted upon loading of the transaction log.

11.5 Increased Scalability of Memory Suballocator Lists

The c-treeACE internal memory suballocator previously utilized 8 lists, each dedicated to a particular range of allocation size and each controlled by a single mutex. An expanded model has been introduced to improve scalability, especially when a large number of alloc or free memory operations takes place at once.

The c-treeACE Server configuration keyword `MEMORY_HASH <N>` (default 1) causes 8N lists to be created, with N dedicated to a particular range of allocation sizes. N is optimally set to the number of CPUs available to the c-treeACE server process.

The `MEMORY_HASH` configuration requires aligned memory boundaries. If a particular server build is compiled without the proper alignment property, a message is logged to *CTSTATUS.FCS* indicating this when this keyword is active:

```
- User# 00001       Configuration error: ctsrvr.cfg, line 3: This c-tree Server does not meet the
compile-time requirements to support the MEMORY_HASH keyword.
```

11.6 Configurable Memory File Hash Bins

The maximum size of a memory file determines the number of hash bins that c-treeACE allocates for the memory file. There is a hard limit on the number of hash bins for a memory file; previously, this limit was set at compile time to 65536.

Now the c-treeACE Server configuration option `MEMFILE_MAX_BINS` can be used to set the maximum number of hash bins allowed for a memory file (default: 65537). Applications storing a large number of records in a memory file should consider increasing this limit.

11.7 Improved Performance Using Unbuffered I/O on Windows Systems

When working with very large cache sizes, it was found that the Windows file system cache could grow to a level that impacted the cache memory usage of the c-treeACE process. On Windows systems, c-treeACE now supports the use of unbuffered disk I/O operations on a per-file basis. Unbuffered I/O bypasses the file system cache and avoids double-caching of data within the server process and the file system cache.

The c-treeACE configuration option `UNBUFFERED_IO` enables unbuffered I/O for the specified file. When the file is opened, additional logic determines the sector size of the disk on which the file resides and stores that information in the new file control block member *dscsiz*.

Windows enforces the following restrictions for I/O when using unbuffered I/O:

1. The file offset for the I/O operation must be a multiple of the disk sector size.
2. The amount of data to be read or written must be a multiple of the disk sector size.
3. The address of the buffer used in the I/O operation must be aligned on a disk sector size boundary.

For files that use unbuffered I/O, c-tree's file I/O function checks that these requirements are met. If not, logic is in place that makes the necessary adjustment by allocating a temporary buffer that is used in the I/O operation.

Example:

```
UNBUFFERED_IO <filename>
```

Use multiple entries to enable unbuffered I/O on multiple files. Wildcard characters are supported.

```
UNBUFFERED_IO custmast.dat
UNBUFFERED_IO *.dat
UNBUFFERED_IO data*
```

Restrictions:

- The use of unbuffered I/O is not supported for encrypted and/or segmented files. If the `UNBUFFERED_IO` configuration option matches the name of a file that uses one of these features, the option is currently ignored.
- Unbuffered I/O is not used during automatic recovery and requests to enable unbuffered I/O on files during automatic recovery are ignored.
- If the sector size of the disk on which the file is stored exceeds the c-treeACE Server page size when unbuffered I/O is requested, the following messages are logged to *CTSTATUS.FCS* and unbuffered I/O is not used on that file:

  An existing file:

```
mbopen: File <filename> disk sector size (<disk_sector_size>) exceeds page size (<page_size>)
```

  A new file:

```
mbcratx: File <filename> disk sector size (<disk_sector_size>) exceeds page size (<page_size>)
```

11.8 Enable Unbuffered I/O for Transaction Logs

Paralleling the unbuffered I/O feature for files, this same technique is also available for transaction logs. `UNBUFFERED_LOG_IO YES` in the c-treeACE configuration file (*ctsrvr.cfg*) enables unbuffered I/O for the transaction logs.

11.9 c-treeACE SQL Statement Cache Tuning Options

c-treeACE SQL statement caches provide a performance benefit by allowing the SQL engine to reuse previously parsed and optimized statements. Proper sizing of this cache can offer significant performance improvements to applications. Two independent caches are used, a static statement cache and a dynamic statement cache. c-treeACE SQL supports setting the size of these statement caches (the number of entries that can be stored) up to 16,000. Previously, the limit was 500 on Windows systems and 800 on Unix systems.

To set the static statement cache size, specify the following configuration option in *ctsrvr.cfg*:

```
SETENV DH_CACHED_STATEMENTS=n
```

To set the dynamic statement cache size, specify the following configuration option in *ctsrvr.cfg*:

```
SETENV DH_DYN_CACHED_STATEMENTS=n
```

On Windows the default size is 150 for each cache and on Unix systems the default size is 250 for each.

Performance Results

FairCom's internal testing demonstrated that setting the static statement cache size to 1,000 dramatically improved the performance of the c-tree load test program, **cttctx**, on a Windows system with 16 CPUs when using *PREIMG* transactions and 16 connections. With the default static statement cache size of 150, the statement cache hit rate was 82% and the transaction rate was 3,700 transactions per second. With the static statement cache size set to 1,000, the statement cache hit rate was 99% and the transaction rate increased to 10,100 transactions per second (nearly three times faster).

The **cttctx** test program executes three SQL statements per file and operates on 23 files, for a total of 69 SQL statements. When more than one SQL thread runs concurrently, multiple copies of a statement can end up in the statement cache. This happens because while a thread is using a SQL statement it removes the statement from the cache. When running the **cttctx** test with 16 connections, we saw up to 16 copies of these 69 statements in the statement cache. As a result, the limit of 150 entries in the static statement cache was much lower than the total likely number of entries used by the 16 connections (which might be 69 * 8 = 552 on average).

FairCom found good performance (around 10,000 transactions per second) in the **cttctx** test with 16 connections when using static statement cache sizes as low as 400. Using a static statement cache size of 300 caused the test to begin with poor performance and after a few seconds it sped up to around 10,000 tps.

11.10 Support Critical Section Spin Count on Windows

While performance testing, we found that modifying the Windows critical section spin count improved performance. The c-treeACE Server for Windows now supports an option to set the spin count for c-treeACE critical sections. The keyword `CRITICAL_SECTION_SPIN <spin_limit>` causes c-treeACE to set the spin count for all its critical sections to the specified value. The critical section spin option defaults to 1000.

A c-treeACE configuration keyword is available to modify this value:

```
CRITICAL_SECTION_SPIN <spin_limit>
```

Only Windows versions supporting the **CriticalSectionAndSpinCount()** function (0x403 or greater) have this feature enabled.

11.11 Reader/Writer Lock Support for Enhanced Performance

Reader/Writer lock support has been added, greatly improving performance with large numbers of clients. Reader/Writer locks are more efficient than a standard mutex by avoiding contention when there are many readers. The server mutex over the memory file hash lists has been changed to use a reader/writer lock when this support is enabled at compile time.

c-treeACE has a generic reader/writer lock implementation available for all platforms. On Windows and Solaris systems, native reader/writer lock synchronization objects are used. When c-treeACE starts, it checks if the system supports reader/writer lock functions and, if so, uses them. If not, it uses generic c-treeACE reader/writer lock functions.

The configuration option `COMPATIBILITY CTREE_RWLOCK` can be used to force c-treeACE to use its reader/writer lock support instead of the native operating system reader/writer lock support.

Look for one of the following messages in *CTSTATUS.FCS* to indicate which, if any, reader/writer lock support is enabled:

```
c-tree reader/writer lock support enabled (keyword specified)
c-tree reader/writer lock support enabled (no O/S support)
Windows native reader/writer lock support enabled
```

See Also

- **Reader/Writer Lock Support for Windows and for Memory Files** (page 144)
- **Reader/Writer Lock Support for Unix Systems** (page 144)

12. Foundations of V10 Performance Gains

FairCom is proud of the unsurpassed performance we have achieved with Version 10. This section summarizes some of the internal changes we've made to make that possible. These enhancements have been performed behind the scenes and will not require you to make any changes to realize the benefits. We include this information for the advanced developer who is interested in understanding the breadth and depth of our effort.

12.1 Transaction Log Improvements

Improvements listed in this section have led to performance gains within the transaction processing subsystem.

Improved Efficiency for Log Templates

An enhanced method of copying transaction log template files is now used improving performance in high transaction rate systems. Previously, a system copy command was executed to copy a transaction log template file (for example, `cp L0000002.FCT L0000002.FCS`). This approach required the full contents of the template file to be read, however, the template file primarily consists of 0xff bytes.

The new approach uses system calls to open the log template file, read its header and write the header to the target log file, then fill a buffer with 0xff bytes and repeatedly write that buffer to the target log file. Configuration options are also available slowing down the copying of the log template in order to reduce the impact of the log template disk I/O write operations on overall c-treeACE Server performance.

Defer Transaction Logging of File Opens

A performance improvement was introduced for applications that perform a large number of file opens without any updates.

The ability to defer assignment of transaction file numbers until a file is updated reduces the number of unnecessary entries written to the transaction log files. Previously, a file opened for update was immediately logged, consuming a temporary file number assigned to the opened file for all of its log entries. But if the file is not updated, there is no benefit to logging its open status as there are no substantive log entries for the file.

The c-treeACE Server now defers the temp file number assignment and logging until the first update is made to a file, resulting in substantial performance gains.

Overlapping Checkpoints

During performance profiling synchronization was improved for a user-invoked checkpoint with an internal checkpoint already in operation. If a checkpoint was forced from an application-level **CTCHKPNT()** call and an in-process checkpoint is already in progress, internal logic has been improved to avoid any additional processing overhead as a checkpoint is already in progress.

Reduced File and User State Variable Contention

During high-performance testing, contention was observed on a global mutex controlling the assignment of transaction numbers and the values of certain transaction state variables that are used by threads other than the owning thread. Logic that cleans up user/file-specific state variables and updates certain file state variables is typically called with this mutex held, which can be quite frequent under heavy loads. This mutex is no longer held during these calls. Further, portions of the logic in this call are now skipped when the function is later called for overall performance gains.

Preimage Savepoint Performance

Preimage space holds key values, data record images, and a host of additional entries that represent pending updates to transaction-controlled files. Each preimage space entry is linked in chronological order as they are added to preimage space.

Preimage space also uses dynamic hash bins to permit efficient searches for data record images. Entries that map to the same hash bin are linked together. To find a record image, the hash bin links are walked until a match is found. This avoids searching the entire chronological linked list.

When a data record is updated across savepoints each version of the record is linked together. This means that each preimage space entry may be on up to three lists: the chronological list, a hash bin list, and a savepoint versions list. Before this modification, a search for a record updated across savepoints would 1) use the hash list to find the first version of the record, and then 2) walk over the chronological links to find the remaining record versions if any.

We now take advantage of the savepoint version list to find the version of the record that corresponds to the current savepoint. This can substantially reduce the number of entries examined in preimage space.

Preimage Search Optimizations

Preimage space holds pending updates on transaction-controlled files and each user has their own preimage space. When a user attempts to read a record, its preimage space must be searched. We now check to see if the calling user has updated the target file during the current transaction. If not, we skip the preimage search, resulting in increased performance.

12.2 Internal Mutex Improvements

A mutex is a synchronization-control object used extensively within the c-treeACE Server. Performance has been improved by minimizing the use of this object. This section details internal mutex changes that have increased throughput.

Shared Memory Logon Contention

The logic has been changed to improve the likelihood of a shared-memory protocol connection attempt succeeding when multiple threads connect at the same time.

Removed Redundant Mutex Usages

An extensive evaluation of how redundant synchronization object calls behaved and interacted with our internal thread impersonation logic was examined. A redundant synchronization call is a call by a thread that succeeds without actually attempting to acquire the synchronization object because the object is already held by the thread. Most importantly, only one call will be made to release the object. Although c-treeACE does not use recursive mutex/semaphore calls, it may allow a mutex/semaphore to be called by the same thread that already owns the synchronization object. We addressed a number of internal details in this area resulting in an overall improvement.

Improve Concurrency of Queue Read and Write Functions

We applied the following changes to reduce the time that the queue write and read functions hold the queue list mutex:

- In the queue write function, memory is now allocated and the data copied to the buffer before acquiring the queue list mutex.
- In the queue read function, the data is now read from the queue entry and the buffer freed for the queue entry after releasing the queue list mutex.

Removed Header Contention for Memory Files

When adding a record to a memory file, the temporary file header mutex was acquired and the buffer that holds the memory record was allocated and initialized. It was determined there is no need for the allocation and initialization to be done under control of file header mutex. The

allocation and initialization of the buffer is now performed outside of the acquisition of the temporary file header mutex, reducing the time held.

Windows Critical Section Spin Count

It was found that modifying the Windows critical section spin count could improve performance. The c-treeACE Server for Windows now supports an option to set the spin count for c-treeACE critical sections. Only Windows versions supporting the **CriticalSectionAndSpinCount()** function (0x403 or greater) have this feature enabled.

For more information, see **Support Critical Section Spin Count on Windows** (page 129).

Mutex Calls Converted to Windows Slim Write Locks at Compile Time

Testing of mutex behavior revealed potential performance improvements when comparing Windows mutex support with Windows slim write locks. The Windows slim write locks perform better in certain situations, so this approach is now used. Further advanced synchronization techniques continue to be explored.

Eliminate Compulsory Atomic Operations for File Block Counters

The increment, decrement, and read of the file block counter is performed while holding a file open/close mutex, and it was found redundant to use atomic increment and decrement. The variable has been changed to a non-atomic variable and the atomic increment and decrement have been changed to non-atomic increment and decrement.

Enhanced Management of Abort Node List Management

Contention was noted involving the abort node list during node splits. When a node that contains key-level locks splits, the key-level locks must be updated for the old node and the new node. Because this affects the abort node list, a loop is called repeatedly to update that list (once for each key-level lock in the node). The intent is to remove the entry from the abort node list, which can be accomplished in one call using a mode we have implemented. This reduces the contention on the mutex that controls the abort node list.

Abort Node Processing Efficiencies During Node Splits

An additional efficiency of the abort node list processing was obtained as the abort node list updates for the after-split nodes are performed without holding a semaphore prone to contention.

12.3 Lock Behavior Improvements

Locking always adds overhead to performance. Any type of improvement, no matter how detailed, can help. The areas of focus are listed in this section.

Commit Read Lock Optimizations

A significant performance improvement has been introduced offering scalability benefits for reads of transaction-controlled files. A commit read lock is a record lock that the c-treeACE Server automatically acquires when reading a record from a transaction-controlled file if the reader does not request a read or write lock on the record. The lock is held while the record is read from c-treeACE Server's data cache.

c-treeACE Server uses a commit read lock to guarantee that the reader reads a consistent version of the record from the data cache. In a performance test involving many clients reading data records without acquiring a lock, we found significant contention on a c-treeACE Server file-specific mutex for commit read lock operations. But a commit read lock is not needed when a data record header and body occupy the same cache page, provided that c-treeACE Server reads the record header and body from the cache in a single call and that c-tree writes the header and body to the cache in a single call.

Now we improve scalability while still guaranteeing record read consistency by reducing the need for commit read locks. c-treeACE Server replaces the separate data record header and body writes to the data cache with a single write to the data cache and only acquires a commit read lock when the data record spans two cache pages. Records that reside in a single cache page are read with no commit read lock overhead. Combining the data record header and body write also improves performance by reducing the number of calls to write to the data cache when updating a record.

Skip Lock Table Entries on Header Lock Calls

A performance improvement was obtained by skipping lock table entries on header lock calls that acquire the header semaphore. While exploring performance bottlenecks, we noticed that on some header calls we always acquire the header semaphore. We do not need the associated lock table entries since the header semaphore provides access control. We now skip the lock table entries, only acquiring header semaphore.

Skip All Locks on Exclusive Opened File

As a continuation of performance-driven lock modifications, all locks on files opened exclusively are now skipped.

Key-Level Lock Processing Optimization

Internal logic has been improved in an internal cleanup routine related to key-level locks. When it is recognized that all the key-level locks in the node belong to the calling thread for the current transaction, cleanup overhead is avoided for these key-level locks. This improves the performance of large transactions that repeatedly add/visit key values during the transaction because unnecessary attempts to cleanup pending key-level locks for the same transaction are no longer attempted.

Faster Partial Record Reads

The ability to read only portions of a data record allows developers to optimize I/O performance. Adjustments were made to ensure that partial record read operations take advantage of the new commit read lock enhancement.

Row-Level Security Filters

Improvements related to the internal handling of row-level filters in conjunction with the new commit read lock technology have been addressed.

12.4 Internal Hash Bin Advancements

Performance gains have been achieved by the improvements in hashing algorithms listed in this section.

Dynamic Hash Logic

A new dynamic hash is used for the transaction log entries held before commit. This has greatly improved efficiency when searching intermediate transaction contents. The hashing logic will dynamically adjust to changing conditions for improved hash management of the transaction and lock data. This permits each individual hash table, independent of the other hash tables, to grow and shrink to balance performance and memory use. The increases and decreases are always a factor of two. That is, if the average number of entries per bin exceeds the load factor, then the number of bins is increased by two. When the number of bins is decreased, it is decreased by a factor of two.

See also **Configure Number of Hash Bins per Memory File** (page 126).

Configurable Memory File Hash Bins

The maximum size of a memory file determines the number of hash bins that c-treeACE allocates for the memory file. There is a hard limit on the number of hash bins for a memory file; previously, this limit was set at compile time to 65536.

Now the c-treeACE Server configuration option `MEMFILE_MAX_BINS` can be used to set the maximum number of hash bins allowed for a memory file (default: 65537). Applications storing a large number of records in a memory file should consider increasing this limit.

Improve Memory File Hash Function for 64-Bit Addresses

The hash function that c-tree uses to assign memory file records to memory file hash bins did not evenly distribute memory records among the hash bins for memory addresses having values over 4 GB. We have implemented an alternative hash function that uses more of the higher-order bits to compute the hash bin.

This new hash function is used on 64-bit systems. We also moved the memory file hash function computation before the call to request the I/O semaphore for the memory file. This might slightly improve performance when more than one client reads/writes a memory file at the same time.

12.5 Memory Related Improvements

The memory-related changes listed in this section have resulted in performance improvements.

Memory Suballocator Enhancements

An expanded model of the c-treeACE internal memory suballocator has been introduced to improve scalability. It is particularly effective when a large number of alloc or free-memory operations take place at once. A server configuration keyword (`MEMORY_HASH <N>`) increases the number of lists used to manage allocation sizes.

For more information, see **Increased Scalability of Memory Suballocator Lists** (page 126).

Improve Memory Suballocator Performance

Memory suballocator performance was improved when searching for a list (known to exist) with available space. The memory suballocator is designed to improve the use of smaller chunks of memory. We found that performance slowed while using a memory file. Profiling suggested a loop in a method that finds an existing “list” of memory objects with one or more available for reuse, or allocates a new list of memory objects was a contention point

When a request for a memory object of a particular type (or size range) is made to the memory suballocator, any available object is first returned (of the same type or size range). If none exists, a new list of memory objects is allocated and the list made the active list. A subsequent request for a memory object is then satisfied from the current active list until all of its memory objects have been used.

Improved list management was implemented allowing quicker searching into multiple lists removing the prior contention points.

Preimage Space Entry Suballocator Lists

We have added support for multiple preimage space entry suballocator lists. Preimage space entries are allocated from a memory suballocator list that is used specifically for allocations of that type. In a multi-threaded performance test we noticed contention on the mutex that serializes access to the preimage memory suballocator list.

Faster Memory Buffer Copies on Unix

c-treeACE buffer management logic was modified on Unix systems to take advantage of the **memmove()** function for right shifts of data in memory. This is when copying data from a smaller

memory address to a larger memory address when the two memory regions might overlap. It is expected that **memmove()** is more efficient than a standard byte-by-byte copy. This approach is already used on Windows systems, and is now extended to Unix platforms.

12.6 API Processing Improvements

The improvements to logic surrounding API calls, including SQL statements, are listed in this section.

Improved Overall Range Performance

Range performance was improved by using **NXTKEY()**/**PRVKEY()** instead of **GTKEY()**/**LTKEY()** when possible.

We discovered that range performance could be enhanced using **NXTKEY()** instead of **GTKEY()** to skip over records that did not meet range and/or filter criteria during range operations. Because **NXTKEY()** cannot always be substituted, the logic now determines when it must use **GTKEY()**.

Improved Open-Ended Key Estimation Performance

The Estimate Key function, **ESTKEY()**, was improved such that if either the upper or lower limit of the range is opened-ended, efficiencies are gained. By open-ended we mean that the estimated number of keys is either greater than a lower limit or smaller than an upper limit.

A NULL pointer is now passed in for the upper or lower limit when this condition is met and the tree manipulations for the open-ended limit can be skipped. The appropriate percentile for the opened-ended limit is simply assigned: 0 for no lower limit or 100 for an opened-ended upper limit.

The tree manipulations consist of a binary search using the First Key routine, **FRCKEY()**, to estimate the percentile of a given limit. To permit some rounding to be accommodated, we actually use a range from 0 to 1000 instead of 0 to 100. Think baseball batting averages.

Improved Performance of Descending Range Searches

We observed unnecessary operations comparing a range performed in ascending order versus descending order. When the key values that satisfy a range are contiguous, the range logic can stop searching whenever an internal get next key operation finds a key that does not satisfy the range criteria. The descending range logic, **PRVRNG()**/**PRVVRNG()**, did not take advantage of this optimization, and made unnecessary calls to low-level key routines.

This modification applies the optimization to the descending order range routines. The key values satisfying a range will be contiguous if:

- All the criteria are equality constraints, *CTIX_EQ*.

  or

- All but the last criteria are equality constraints and the last of the criteria does not use a NOT condition (i.e., it is not a *CTIX_NE* or *CTIX_NOTBET*).

Trailing segments, if any, that have no range criteria do not affect whether or not the keys satisfying the range criteria are contiguous.

Persisted Current Index Buffer Node for NXTKEY and PRVKEY Operations

The **NXTKEY()** and **PRVKEY()** functions already remember the offset of the current index node so that subsequent calls can continue scanning from the previous key position. Now, c-treeACE remembers the address of the index buffer that holds the current node.

Whenever the current index node offset is saved for later use, the current index buffer address is also saved. When **NXTKEY()** or **PRVKEY()** find that the current index buffer address has been set, these functions lock that index buffer and then verify that the buffer still contains the same node. If so, the function continues processing using the locked index buffer. If not, the function calls the node management routines as usual, and remembers the address of the returned node buffer.

Set Node Name Optimization

The set node name function, **SETNODE()**, was optimized to take no action if the specified node name is same as current node name. This avoids a client network call to the c-treeACE Server and avoids an entry written to the transaction log buffer.

Key-Level Lock logic

Performance was enhanced when the calling thread owns all the pending key-level locks.

In some situations repeated calls were made to a function which attempts to resolve key-level locks without any cleanup occurring. This cleanup function checks each of the key-level locks in a leaf node to determine if the associated transactions have terminated. When a key-level lock is associated with a transaction that has been committed or aborted, the key-level lock is removed and the key value stays (committing an add or aborting a delete) or is removed (committing a delete or aborting an add). But if a thread is executing a long transaction, it may be repeatedly calling this cleanup routine without any effect when the key-level locks for a node all belong to the calling thread.

We now immediately detect when all the key-level locks belong to the calling thread, and return without checking each of the key-level locks. Also, we have enhanced the key-level locking cleanup performance when calling thread is executing a **NXTKEY** operation. To improve the efficiency of **ctclup()** when it is called as part of a **NXTKEY()** operation, **ctclup()** has been modified to only perform the cleanup necessary to satisfy the **NXTKEY** operation.

Improve Performance of Range Retrieval

An enhancement improves the performance of range retrieval with equality search on a full unique key. When a range search is performed on a unique index and the range criteria specify an equality match on all segments of the key, we can call **EQLKEY()** instead of **GTEKEY()**. For a partitioned file global-unique index that does not cover the partition key, this greatly improves performance when many active partitions exist. This is because **EQLKEY()** can use the global unique partition host index to find the partition that contains the key value.

Prior to this modification, the range search function called **GTEKEY()**, which required calling **GTEKEY()** on each partition and returning the minimum key value found.

12.7 I/O Performance Improvements

This section lists improvements in I/O that have resulted in performance enhancements.

Permit Physical ISAM Files to Remain Open

An enhancement now permits physical ISAM files to remain open even after all users have closed them. When a non-memory file is physically closed, that is, all users have closed the file for their use, c-treeACE normally removes the data cache and/or index buffer entries associated with the file. An ability to “hold” the file open would retain the cache contents for quicker retrieval on re-open.

Memory files already employ the option of staying open after all users have closed them to avoid losing the contents of the file such that their contents can be accessed by subsequent opens. This *KEEPOPEN* feature is now extended to non-memory files. With this, physical ISAM data and index files can be designated as *KEEPOPEN* files, and retain their data cache and index buffers. It also eliminates a physical open when the next user opens the file. The **ctCLSNAM()** API call can be made to finally physically close the files when they no longer require their contents cached.

Support Unbuffered I/O on Windows Systems

The c-treeACE Server on Windows now supports the use of unbuffered disk I/O operations on a per-file basis. Unbuffered I/O bypasses the file system cache provided by the operating system.

As c-treeACE already maintains a sophisticated cache mechanism, system resources can be minimized by not utilizing the operating system file system cache. This avoids double caching of data and lowers the memory demands of the operating system because both caches are not required.

For more information, see **Improved Performance Using Unbuffered I/O on Windows Systems** (page 127).

Enable Unbuffered I/O for Transaction Logs

Paralleling the unbuffered I/O feature for files, this same technique is also available for transaction logs. `UNBUFFERED_LOG_IO YES` in the c-treeACE configuration file (*ctsrvr.cfg*) enables unbuffered I/O for the transaction logs.

Efficient Key Count Updates for Transaction Controlled Files

Enhanced performance of transaction controlled files has been gained by avoiding header lock/unlock when updating the key count in the file header. Header update logic updates the key count and other information for an index file. However, this is not necessary for a transaction-controlled index file as updated user-specific state information is correctly applied to the file at transaction commit.

The index header lock and unlock calls were removed for transaction-controlled files and during automatic recovery. This benefits performance and scalability of index key insert and delete operations on a transaction-controlled index file by reducing the header update contention.

Skip Unnecessary Read of VARLEN Record Headers

Unnecessary reads of variable length record headers are now avoided during record read operations. It was noticed that variable-length record headers were read twice. Specific logic checked if the record header was marked active and that the used record length was non-zero. However, these conditions are appropriately checked elsewhere. Skipping the additional check results in optimized reads.

Extended Header Write Optimization

We now write both the primary and extended headers in a single I/O operation for a file. This greatly improves performance, notably for *WRITHRU* files.

Reader/Writer Lock Support for Windows and for Memory Files

Using an operating system's native support for reader/writer locks is more efficient than using a standard mutex. This is especially advantageous because it avoids contention when there are many readers. We have implemented reader/writer lock support for Windows systems, and we changed the mutex on the memory file hash lists to a reader/writer lock. See **Reader/Writer Lock Support for Enhanced Performance** (page 129).

Reader/Writer Lock Support for Unix Systems

In addition to the implementation of reader/writer locks synchronization objects on Windows, support was added for a reader/writer lock synchronization object on c-treeACE Servers on Unix systems using the native pthread library reader/writer lock support. On Solaris SPARC systems we continue to use Solaris' native reader/writer lock support. See **Reader/Writer Lock Support for Enhanced Performance** (page 129).

File Name Hash List

Improvements to the file name hash list speed up file opens when many files are already opened. A file name hash list was introduced that is checked when a file is being opened to see if the file is already open. A file's name is put into the hash list when the file is created or opened and is removed from the hash list when the file is physically closed.

With this strategy, optimization and execution of a SQL query that opened over 5000 files (multiple partitioned files) decreased from 10 seconds to 3 seconds on a specific test case.

12.8 Profiling Results/Processing Improvements

Speed is always crucial and we routinely profile our code to squeeze out every advantage we can find. The areas listed in this section were improved in this release.

COMMIT_DELAY Value Modified for Windows Systems

Our experience with Windows shows that using `COMMIT_DELAY` can dramatically improve performance of *TRNLOG* transactions. This default is now `COMMIT_DELAY 2` for Windows systems.

Faster Internal User File Control Block Reallocation

When a key search occurs on partitioned indexes that are not ordered as the partition key, all the partition members must be opened, which can take a significant amount of time if the number of partitions is large. One operation where unnecessary time was spent was resizing (allocate new + copy + free old) a user's file control block. Previously, this grew in increments of 32 (MAXMEMB + 1) files.

With partitioned files accessed via SQL, thousands of files may be opened by a single query. This change increases the rate at which this array grows by doubling in size up to 2,000 files and reduces the total query time by 4% in a case with 1,000 files.

Reduce 64-Bit Arithmetic Calls When a Non-Huge File Is Processed

We observed a significant processing burden in a performance test in calls to the 64-bit arithmetic routines in the function that manages the data file I/O through the data cache. For members of superfiles, it is also possible for index header writes and space management node writes for the "index" I/O to ultimately get to disk through this same data cache processing code path.

This code was modified to skip the 64-bit arithmetic routines for non-huge files, and simply perform 32-bit arithmetic on the low order file addresses. We also observed a file address computation that could be removed completely for both huge and non-huge files.

Dynamic Dump Thread in Potential Livelock

The internal I/O polling loop now checks to see if a read is attempted by the dynamic dump thread while the segment-updating thread is waiting in the polling loop. If so, the dynamic dump thread performs its read permitting it to make progress on clearing the dynamic dump indicator, and eventually releasing the segment-updating thread.

Adaptive Defer Loops for Better Index Buffer Performance

Performance testing found that modifying the default 10-millisecond defer time in the mutex controlling index buffer retrievals substantially increased performance. In our testing, the transaction rate increased from 6686 transactions per second (tps) to 13748 tps—a twofold increase. The retry logic in the defer loop now increases 1 millisecond for each three spins of the loop until a maximum of 10 milliseconds to avoid unnecessary starvation of resources.

Improved File Number Search Efficiency

Internal file numbering management either looks for a block of available file numbers or looks to see if a given file number heads up a sufficiently large block of available file numbers. An opportunity was found to improve the loop that steps over file numbers looking for a block of file numbers. Instead of stepping through the loop sequentially one file number each time, those file numbers that cannot possibly work because a particular file number is unavailable are skipped, and will be unavailable for all starting file numbers up to and including this particular file number.

Improved Index Node Access Performance

Performance analysis revealed that index node management was performing many retries (that is, attempting to find an index buffer with the desired node) when the expected node was already in a buffer. This improvement avoids retries when the node holds the desired buffer and state and skips the queue.

A. Function Reference

This appendix summarizes the function reference for several of the new features added in c-treeACE V10:

You will find a complete function reference, containing many more API calls, at www.FairCom.com.

A.1. Data Compression API

This section describes the data compression functions:

ctSetCompress

Sets the data compression type and attribute to be used when creating files with data compression.

Note: To enable data compression, you need to either use the `COMPRESS_FILE` keyword or programmatically add *ctCompressRec* in the *XCREblk.splval* (for ISAM and Low-Level API).

Declaration

```
ctSETCOMPRESS( NINT comptype, NINT compvrsn, pTEXT dllname, pTEXT attrstr, VRLEN attrlen )
```

Description

- *comptype* - either *ctCMPRECzlib (default)*, *ctCMPRECsRLE, ctCMPRECuser (user supplied shared library)*
- *compvrsn* - alternative versions for a given type
- *dllname* - required when *comptype*==*ctCMPRECuser*

  attrstr - NULL if no fine-tuning parameters are specified, or the following optional parameter format is:

  "parm1=value; ... parmn=value;"

  zlib specific parameter names are:
  - *level*
  - *strategy*
  - *windowbits*
  - *memlevel*
  - *method*

  sRLE specific parameter names are:
  - *level [0 - 7]*
  - *strategy [0]*
- *attrlen* - combined length of the *attrstrf* strings including the null-termination bytes

Similar in operation to **ctSETENCRYPT()**, **ctSETCOMPRESS()** determines the compression specifics that will be used for all compressed data files created after the call to **ctSETCOMPRESS()**. To use different compression routines for different files, the application creating the files must call **ctSETCOMPRESS()** before each of the creates. Calling **ctSETCOMPRESS()** does not affect the compression used by existing files. When existing files are opened the compression resource is used to specify how the records are compressed. Such an open will fail if the resource contains a DLL name that is not available.

If no call is made to **ctSETCOMPRESS()**, then default, built-in zlib compression routines are used, as long as no server configuration is specified otherwise.

If no DLL is required, *dllname* is NULL. If no fine-tuning parameters are specified, then *attrstrf* is NULL and *attrlen* is zero. (See **User Defined Compression Dynamic Shared Library** (page 152) for additional details.)

Notes

- ctCompressRec is only allowed for data files. An error 944 (**AREC_MOD**) will be generated if it is used in the XCREblk corresponding to an index file.

- If a data file to be compressed is fixed-length, it will be changed to a variable-length file at create time. This means there will be an additional 10 bytes of information for each record in the file for the variable-length record management (as discussed in *Variable-Length Records* (https://docs.faircom.com/doc/*ctreeplus/62574.htm*)). Therefore, be sure compression makes sense on a fixed-length file before turning it on.
- This feature is available only for ISAM files fully maintained by ISAM updates. Low-Level c-treeACE functions will return an error when used on files with the compression attribute enabled.
- Some files compress better than others (text data vs. binary data for instance). Initial testing revealed up to an 80:1 compression ratio in some cases.
- Additional compression routines may become available in the future.

Server Configuration

Default c-treeACE compression can be specified with the following configuration keywords:

```
CMPREC_TYPE          < "ZLIB" | "USER" | "RLE" >
CMPREC_VERSION               <a number >= 1>
CMPREC_DLL           <name of DLL>
```

These keywords should be entered in the configuration file in the order shown, and a DLL name is required for `CMPREC_TYPE` of USER.

The `COMPRESS_FILE` keyword can be used to enable compression in files whose names match specified file names (including wildcards).

Run Length Encoding (RLE) compression option is available in V10.3 and later. Configure the RLE compression algorithm in *ctsrvr.cfg*, similar to using the keywords ZLIB or USER.

In v10.3.0 and later, c-tree on Unix/Linux platforms dynamically loads zlib instead of statically linking with it.

Compressed Files in FairCom DB API - In V10 and later, the FairCom DB API file create mode, `CTCREATE_COMPRESS`, enables compressed record support in FairCom DB API. When `CTCREATE_COMPRESS` mode is used, FairCom DB API automatically creates the file as variable-length.

Return

AREC_BUF	936	Could not allocate a buffer for an augmented variable-length record. (for example, *ctCMPREC*).
AREC_SUP	937	Augmented variable-length records not supported for *SUPERFILE* host, fixed length files or index files.
AREC_DCM	938	Could not decompress the data record. *sysiocod* set to Z_xyz_ERROR.
AREC_ISM	939	Compressed record-length files require ISAM update; not Low-Level update.
AREC_NOP	940	Requested operation not supported for augmented records.

AREC_MOD	944	Bad augmented record file mode at create. Two conditions return this error: • Setting ctCompressRec in the XCREblk corresponding to an index file. It is only allowed for data files. • Setting ctCompressRec on a data file that is not variable-length (ctVLENGTH).

See also

User Defined Compression Dynamic Shared Library (page 152)

User Defined Compression Dynamic Shared Library

The c-treeACE data record compression feature supports user definable compression modules. To include a custom compression module, the following information must be made available from within the shared library.

Function Arguments

`pCMPRECFNC pfnc;` input pointer to function pointer/DLL structure defined in *ctstrc.h.*

`ppVOID pattrbuf;` on input, a pointer to optional, fine-tuning parameter buffer. **pattrbuf*==NULL implies no fine-tuning parameters specified. On output, **pattrbuf* points to a parameter buffer. **pattrbuf* is not guaranteed to be NULL on output if it is NULL on input, and vice versa.

`pVRLEN pattrlen;` on input, pointer to length of fine-tuning parameter buffer. If **pattrbuf* is NULL on input, **pattrlen* must be zero. On output **pattrlen* contains the length of the parameter buffer.

`pCMPRECRES pres;` input pointer to compression resource structure defined in *ctstrc.h.*

`pVOID context;` input pointer to optional, compression/decompression support structure. If the compression (*CmpActn*) and/or decompression (*ExpActn*) routines do not use a support structure, then context will be NULL or *((pVOID) 1)*.

`pVOID source;` input pointer to byte stream to be compressed or decompressed.

`VRLEN slen;` input length of source buffer.

`pVOID destination;` output pointer to buffer that will contain the result of compression/decompression.

`pVRLEN pdlen;` on input, **pdlen* is length of destination buffer. On output, **pdlen* is the length of the byte stream result from the compression/decompression.

Function Prototypes

The module *ctzlib.c* has implementations of these functions for the built-in zlib support.

FncInit

```
NINT FncInit(pCMPRECFNC pfnc,pTEXT attrstr,VRLEN attrlen)
```

This routine performs a one-time setup on each physical open of a file after the DLL has been loaded. *attrstr* and *attrlen* are the same as those in **ctSETCOMPRESS()**. In addition to any DLL specific requirements, **FncInit()** should verify that the compression type and version (*pfnc->comptype* and *pfnc->compvrsn*) are compatible with the DLL, and that the fine-tuning parameters, if any, are valid. If no fine-tuning parameters are passed in, then if the routines

require a parameter set, **FncInit()** should allocate and initialize such a parameter buffer. If it does so, then it should set *pfnc->allocbuf* to the address of the allocated buffer.

FncInit() returns NO_ERROR or **CMPR_ERR** (). In the case of **CMPR_ERR** **FncInit()** should set the *sysiocod* (using the routine *usys(code)*) as listed in *cterrc.h* for **CMPR_ERR**. **FncInit()** does not actually set *uerr_cod*.

FncExit

```
NINT  FncExit(pCMPRECFNC pfnc )
```

FncExit() is called just before a file is closed and its associated DLL is unloaded. In addition to DLL specific requirements, **FncExit()** should free memory allocated for a parameter buffer. **FncExit()** can test *pfnc->allocbuf* to find the allocated buffer address, if any.

FncExit() should return NO_ERROR.

CmpInit

```
pVOID CmpInit(pCMPRECFNC pfnc)
```

CmpInit() is called each time a user opens a compressed data file. **CmpInit()** should allocate a context buffer if the DLL requires one, and initialize the context as needed. The context buffer would typically be used to initialize and maintain the compression state. If no such context buffer is needed, then return *((pVOID) 1)*.

CmpInit() returns the address of the context buffer, or *(pVOID) 1* if no context is used, or NULL on error. In case of error call *usys(35)* to indicate the **CmpInit()** failure.

CmpActn

```
NINT  CmpActn(pVOID context, pCMPRECFNC pfnc, pVOID source, VRLEN slen, pVOID destination, pVRLEN pdlen)
```

CmpActn() performs the actual compression of the source into the destination. *pdlen holds the length of the destination buffer on input, and is set to the length of the compressed output on return. CmpActn should be aware that an input value of *((pVOID) 1)* means that no context structure has been provided.

CmpActn() returns **CMPREC_OK** () on success, **CMPREC_BUF** () if the destination buffer is too small, or **CMPR_ERR** for an unexpected error. In case of an unexpected error, *usys(1000 + err)* should be called to set *sysiocod* where *err* is the internal error code used by the compression routines. (See **zlib1_CmpActn()** for an example.)

CmpExit

```
NINT  CmpExit(pVOID context, pCMPRECFNC pfnc)
```

CmpExit() is called each time a user closes a compressed data file. **CmpExit()** should de-initialize a context buffer, if one has been allocated in **CmpInit()**, and free the associated memory.

CmpExit() typically returns NO_ERROR unless the DLL specific code to de-initialize the context buffer returns an error code in which case return this error code.

The following three functions behave just as their compression counterparts except that they apply to decompression. **ExpActn()** has the same return values as **CmpExp()** except that attempting to decompress corrupted data results in a return value of *CMPREC_DATA* and *usys(2000 + err)* is called where *err* is the DLL specific internal code signifying the corruption.

```
pVOID ExpInit(pCMPRECFNC pfnc)
NINT  ExpActn(pVOID context, pCMPRECFNC pfnc, pVOID source, VRLEN slen, pVOID destination, pVRLEN pdlen)
NINT  ExpExit(pVOID context, pCMPRECFNC pfnc)
```

A.2. Virtual Table API

This section describes the FairCom DB API functions available to enable Virtual Table support:

ctdbAddMRTTable

ctdbAddMRTTable() adds an existing *MRTTable* definition to the database dictionary.

Declaration

```
CTDBRET ctdbAddMRTTable(pCTDBDATABASE pDatabase, pTEXT Name, pTEXT ParentName, UINT info);
```

Description

- *pDatabase*: the database handle
- *Name*: the *MRTTable* name
- *ParentName*: the host table name
- *info*: the *MRTTable* definition number

ctdbAddMRTTable() functions much as **ctdbAddTable()** in adding an existing table to the database dictionary.

Return Values

Value	Symbolic Constant	Explanation
0	CTDBRET_OK	Successful operation.

See c-tree Plus Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid c-tree Plus error values.

Example

Here is example pseudo code to add an existing *MRTTable* to a database dictionary. This example also demonstrates the **ctdbAllocVTableInfo() ctdbGetVTableInfoFromTable()**, and **ctdbAddMRTTable()**, **ctdbFreeVTableInfo()** functions.

For this example, assume that a *MRTTable* is already defined and you copy the host table to another machine. In this case, you simply need to add the tables to the new database dictionary.

```
pCTDBVTABLEINFO VTableInfo;
pCTDBTABLE hTable;

hTable = ctdbAllocTable(hDatabase);
if (hTable == NULL)
{
    /* error */
}

retval = ctdbOpenTable(hTable, "tutorial_host", CTOPEN_EXCLUSIVE);
if (retval != CTDBRET_OK)
{
    /* error */
}

VTableInfo = ctdbAllocVTableInfo(hTable, 200); /*retrieve max 200 definition (which is the maximum
supported) */
if (VTableInfo == NULL)
{
    /* error */
}

ctdbGetVTableInfoFromTable(VTableInfo);
ctdbCloseTable(hTable);

for (i = 0; i < VTableInfo->actual_elements; i++)
{
    retval = ctdbAddMRTTable(hDatabase, VTableInfo->data[i].name, "tutorial_host",
VTableInfo->data[i].id);
    if (retval != CTDBRET_OK)
    {
            /* error */
    }
}
ctdbFreeVTableInfo(VTableInfo );
```

See Also

ctdbAllocVTableInfo(), **ctdbCreateMRTTable()**, **ctdbFreeVTableInfo()**, **ctdbGetVTableInfoFromTable()**, **ctdbGetVTableNumber()**, **ctdbIsVTable()**, **ctdbRemoveVTableResource()**, **ctdbSetMRTTableFilter()**

ctdbAddVTableResource

ctdbAddVTableResource() adds a virtual table resource.

Declaration

```
CTDBRET ctdbAddVTableResource(CTHANDLE Parent, CTHANDLE Child)
```

Description

- *Parent*: Host table to which the virtual table is assigned.
- *Child*: Virtual table handle.

Note: This function does not update the database dictionary and does not support callbacks. This function is used internally by FairCom DB API and is documented only for completeness.

Return Values

Value	Symbolic Constant	Description
4130	**CTDBRET_NOMOREVTRES**	No more RESOURCE for vtable available on parent table.
4131	**CTDBRET_VTABLEEXIST**	The table has *VTable* defined in the dictionary.
4132	**CTDBRET_VTABLETYPE**	The *VTable* type in the dictionary mismatches the one in the resource.

See c-tree Plus Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid c-tree Plus error values.

See Also

ctdbAddMRTTable(), **ctdbAllocVTableInfo()**, **ctdbCreateMRTTable()**, **ctdbFreeVTableInfo()**, **ctdbGetVTableInfoFromTable()**, **ctdbGetVTableNumber()**, **ctdbIsVTable()**, **ctdbRemoveVTableResource()**, **ctdbSetMRTTableFilter()**

ctdbAllocVTableInfo

ctdbAllocVTableInfo() allocates a new *CTDBVTABLEINFO* structure.

Declaration

```
CTHANDLE ctdbAllocVTableInfo(pCTDBTABLE pTable, UCOUNT size);
```

Description

- *pTable*: Table to allocate a *CTDBVTABLEINFO* object
- *size*: Number of elements allocated

Return Values

ctdbAllocVTableInfo() returns a FairCom DB API handle to a *CTDBVTABLEINFO* structure.

See Also

ctdbAddMRTTable(), **ctdbAddVTableResource()**, **ctdbCreateMRTTable()**, **ctdbFreeVTableInfo()**, **ctdbGetVTableInfoFromTable()**, **ctdbGetVTableNumber()**, **ctdbIsVTable()**, **ctdbRemoveVTableResource()**, **ctdbSetMRTTableFilter()**

ctdbCreateMRTTable

ctdbCreateMRTTable() creates a new *MRTTable* based on a host table.

Declaration

```
CTDBRET ctdbCreateMRTTable (CTHANDLE Handle, pTEXT VTableName, pTEXT ParentName, CTCREATE_MODE CreateMode,
pTEXT filter);
```

Description

- *Handle*: *MRTTable* handle
- *VtableName*: name of the *MRTTable*
- *ParentName*: name of the host (or parent) table
- *CreateMode*: virtual table create mode
- *filter*: the filter to apply to the host table to identify the record belonging to this *MRTTable*

Once created, an *MRTTable* behaves as a regular FairCom DB API table.

Limitations: At this time it is not possible to use record filters and the **ctdbAlterTable()** function on a *MRTTable* Table.

Return Values

Value	Symbolic Constant	Description
4130	**CTDBRET_NOMOREVTRES**	No more RESOURCE for VTable available on parent table.
4131	**CTDBRET_VTABLEEXIST**	The table has *VTable* defined in the dictionary.
4132	**CTDBRET_VTABLETYPE**	*VTable* type mismatch between dictionary and resource.

See c-tree Plus Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid c-tree Plus error values.

Example

```
CTHANDLE pField0, pField1, pField2, pField3, pField4;
pField0 = ctdbAddField(hTableCustOrdr, "rectype", CT_FSTRING, 1);
pField1 = ctdbAddField(hTableCustOrdr, "co_ordrnumb", CT_FSTRING, 4);
pField2 = ctdbAddField(hTableCustOrdr, "co_ordrdate", CT_DATE, 4);
pField3 = ctdbAddField(hTableCustOrdr, "co_promdate", CT_DATE, 4);
pField4 = ctdbAddField(hTableCustOrdr, "co_custnumb", CT_FSTRING, 4);

if (!pField0 || !pField1 || !pField2 || !pField3 || !pField4)
    Handle_Error("Define(): ctdbAddField()");

/* create table */
if (ctdbCreateMRTTable(hTableCustOrdr, "custordr","tutorial_host",
CTCREATE_NORMAL|CTCREATE_NONULFLD,"rectype==\"O\""))
    Handle_Error("ctdbCreateMRTTable()");
```

Note:

In V11.5, there has been a ctdbCreateMRTTable behavior change.

See Also

ctdbAddMRTTable(), **ctdbAddVTableResource()**, **ctdbAllocVTableInfo()**, **ctdbFreeVTableInfo()**, **ctdbGetVTableInfoFromTable()**, **ctdbGetVTableNumber()**, **ctdbIsVTable()**, **ctdbRemoveVTableResource()**, **ctdbSetMRTTableFilter()**

ctdbFreeVTableInfo

ctdbFreeVTableInfo() frees memory associated with a *CTDBVTABLEINFO* structure.

Declaration

```
VOID ctdbFreeVTableInfo(pCTDBVTABLEINFO Info);
```

Description

- *Info*: A *CTDBVTABLEINFO* object

Return Values

ctdbFreeVTableInfo() does not return a value.

See Also

ctdbAddMRTTable(), **ctdbAddVTableResource()**, **ctdbAllocVTableInfo()**, **ctdbCreateMRTTable()**, **ctdbGetVTableInfoFromTable()**, **ctdbGetVTableNumber()**, **ctdbIsVTable()**, **ctdbRemoveVTableResource()**, **ctdbSetMRTTableFilter()**

ctdbGetVTableInfoFromTable

ctdbGetVTableInfoFromTable() lists virtual tables defined in a host file.

Declaration

```
VOID ctdbGetVTableInfoFromTable(pCTDBVTABLEINFO VtableRes)
```

Description

There may be a need to list the virtual tables defined in a host file without looking at the FairCom DB API database dictionary (for example, the dictionary is missing). This information can be reconstructed with this function.

- *pCTDBVTABLEINFO VtableRes*: a pre-allocated structure (using **ctdbAllocVTableInfo()**) to contain information retrieved from the table resources.
  - *VtableRes->hTable*: the host table handle (*pCTDBTABLE*)
  - *VtableRes->data*: an array of *tagVTABLEOBJ*
    - *data->id*: the virtual table number (*UCOUNT*)
    - *data->type*: the virtual table type (*VTABLE_TYPE*)
    - *data->name*: the virtual table name (*pTEXT*) (when applies)
  - *VtableRes->data_elements*: the maximum number of elements *data* could contain (*UCOUNT*)
  - *VtableRes->actual_elements*: the actual number of elements set in *data* (*UCOUNT*)

The *hTable* and *data_elements* fields are populated by **ctdbAllocVTableInfo()**. *data* and *actual_elements* are populated by **ctdbGetVTableInfoFromTable()**.

Note: ctdbFreeVTableInfo() should be used in order to properly free memory allocated for *CTDBVTABLEINFO*.

Return Values

ctdbGetVTableInfoFromTable() does not return a value.

See Also

ctdbAddMRTTable(), **ctdbAddVTableResource()**, **ctdbAllocVTableInfo()**, **ctdbCreateMRTTable()**, **ctdbFreeVTableInfo()**, **ctdbGetVTableNumber()**, **ctdbIsVTable()**, **ctdbRemoveVTableResource()**, **ctdbSetMRTTableFilter()**

ctdbGetVTableNumber

ctdbGetVTableNumber() retrieves the number assigned to a virtual table.

Declaration

```
UINT ctdbGetVTableNumber(CTHANDLE table)
```

Description

- *table*: A handle to any FairCom DB API table.

Upon a virtual table open, FairCom DB API internally opens the virtual table host and loads the definition of the virtual table from a resource contained in the host table. To identify among multiple possible virtual tables created on a single host table, a definition number is used which is automatically assigned by FairCom DB API when defining the virtual table and is stored in the database dictionary.

The dictionary entry for a virtual table contains (in addition to existing table information) links to the host table and the virtual table number.

The resource containing the virtual table information has a resource type set to *FC_CTDB_VTABLES* (2). Resource numbers range from *FCRES_CTDB_VTABLES* (0) to *FCRES_CTDB_VTABLES_END* (199). Its content is as follows (packed, no alignment...)

- First byte: endian-ness
- Next 4 bytes: virtual table type

The remainder of the resource is variable and depends on the virtual table type.

Return Values

ctdbGetVTableNumber() returns the number assigned to a virtual table.

See Also

ctdbAddMRTTable(), **ctdbAddVTableResource()**, **ctdbAllocVTableInfo()**, **ctdbCreateMRTTable()**, **ctdbFreeVTableInfo()**, **ctdbGetVTableInfoFromTable()**, **ctdbIsVTable()**, **ctdbRemoveVTableResource()**, **ctdbSetMRTTableFilter()**

ctdbIsVTable

ctdbIsVTable() can be used to check if a table is a virtual table or a regular table.

Declaration

```
CTBOOL ctdbIsVTable(CTHANDLE table)
```

Description

- *table*: A handle to any FairCom DB API table.

Return Values

ctdbIsVTable() returns *YES* if the table is a Virtual Table, *NO* otherwise.

See Also

ctdbAddMRTTable(), **ctdbAddVTableResource()**, **ctdbAllocVTableInfo()**, **ctdbCreateMRTTable()**, **ctdbFreeVTableInfo()**, **ctdbGetVTableInfoFromTable()**, **ctdbGetVTableNumber()**, **ctdbRemoveVTableResource()**, **ctdbSetMRTTableFilter()**

ctdbRemoveVTableResource

ctdbRemoveVTableResource() removes a virtual table resource.

Declaration

```
CTDBRET ctdbRemoveVTableResource(CTHANDLE Parent, NINT number)
```

Description

- *Parent*: Host table to which the virtual table is assigned.
- *number*: Number of the virtual table.

Note: This function does not update the database dictionary and does not support callbacks. This function is used internally by FairCom DB API and is documented only for completeness.

Return Values

Value	Symbolic Constant	Description
4130	**CTDBRET_NOMOREVTRES**	No more RESOURCE for VTable available on parent table.
4131	**CTDBRET_VTABLEEXIST**	The table has *VTable* defined in the dictionary.
4132	**CTDBRET_VTABLETYPE**	The *VTable* type in the dictionary mismatches the one in the resource.

See c-tree Plus Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid c-tree Plus error values.

See Also

ctdbAddMRTTable(), **ctdbAddVTableResource()**, **ctdbAllocVTableInfo()**, **ctdbCreateMRTTable()**, **ctdbFreeVTableInfo()**, **ctdbGetVTableInfoFromTable()**, **ctdbGetVTableNumber()**, **ctdbIsVTable()**, **ctdbSetMRTTableFilter()**

ctdbSetMRTTableFilter

ctdbSetMRTTableFilter() sets filter information in the virtual table handle.

Declaration

```
CTDBRET ctdbSetMRTTableFilter(CTHANDLE Handle, pTEXT condition)
```

Description

When implementing virtual table callbacks the **ctdbSetMRTTableFilter()** function may be of use.

This function sets the filter information in the *MRTTable Handle* such that at the end of the open procedure FairCom DB API knows which filter to use.

Return Values

Value	Symbolic Constant	Description
4130	**CTDBRET_NOMOREVTRES**	No more RESOURCE for vtable available on parent table.
4131	**CTDBRET_VTABLEEXIST**	The table has *VTable* defined in the dictionary.
4132	**CTDBRET_VTABLETYPE**	The *VTable* type in the dictionary mismatches the one in the resource.

See c-tree Plus Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid c-tree Plus error values.

See Also

ctdbAddMRTTable(), **ctdbAddVTableResource()**, **ctdbAllocVTableInfo()**, **ctdbCreateMRTTable()**, **ctdbFreeVTableInfo()**, **ctdbGetVTableInfoFromTable()**, **ctdbGetVTableNumber()**, **ctdbIsVTable()**, **ctdbRemoveVTableResource()**

A.3. Sequence Number API

c-treeACE V10 introduces a generic method to create, maintain, and access multiple sources of sequential numbers through a new persisted **Sequence** feature.

ctCreateSequence

Declaration

```
NINT ctCreateSequence(pctSEQATTR pseqattr);
```

Description

Creates a new sequence with the specified attributes. The following *ctSEQATTR* structure fields must be initialized:

- *seqnam* - The name of the new sequence. No sequence by that name must currently exist.
- *seqini* - The initial sequence value. If the sequence enforces a limit, the initial sequence value must be less than the sequence limit for an incrementing sequence or must be greater than the sequence limit for a decrementing sequence.
- *seqcur* - The current sequence value. For an incrementing sequence, the current sequence value must be greater than or equal to the initial sequence value, and if the sequence enforces a limit the current sequence value must be less than or equal to the limit value. For a decrementing sequence, the current sequence value must be less than or equal to the initial sequence value, and if the sequence enforces a limit the current sequence value must be greater than or equal to the sequence limit. If *sequnk* is set to a non-zero value, the *seqcur* value is ignored, and the current sequence value is set to the unknown value.
- *seqinc* - The sequence increment/decrement amount: It must be a positive value that is less than the difference between the initial sequence value and the sequence limit
- *seqlim* - The sequence limit: The sequence limit must be greater than the initial sequence value for an incrementing sequence or must be less than the initial sequence value for a decrementing sequence. The sequence limit is only enforced if the sequence type specifies the *ctSEQLIM* bit.
- *seqtyp* - The sequence type: It must be set to one of:
  - *ctSEQINC* (incrementing sequence)
  - *ctSEQDEC* (decrementing sequence)

  and either one of:
  - *ctSEQCYC* (cycling sequence)
  - *ctSEQTRM* (terminating sequence)

  Optionally include *ctSEQLIM* (enforce sequence limit).
- *sequnk* - Setting the sequence unknown flag causes the initial sequence value to be set to the unknown value. Otherwise, the current sequence number is set to the initial sequence number.

Return Values

Value	Symbolic Constant	Explanation
0	**NO_ERROR**	Successfully created the sequence.
900	**SEQDUP_ERR**	A sequence having the specified name already exists.

Value	Symbolic Constant	Explanation
903	**SEQTYP_ERR**	The specified sequence type contains an invalid combination of sequence type options.
904	**SEQINI_ERR**	The initial value specified for the sequence is out of range.
905	**SEQCUR_ERR**	The current value specified for the sequence is out of range.
906	**SEQLIM_ERR**	The limit value specified for the sequence is out of range.
907	**SEQINC_ERR**	The increment value specified for the sequence is out of range.
901	**SEQNAM_ERR**	An invalid sequence name was specified: the name is NULL, empty, or too long.

See c-treeACE Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid error values.

Example

```
ctSEQATTR   seqattr;
    NINT            rc;
    /*
    ** Create an incrementing sequence that starts with 1, increments by 3, and
    ** terminates with 100.
    */
    strcpy(seqattr.seqnam, "MyFirstSequence");
    seqattr.seqini = 1;
    seqattr.seqinc = 3;
    seqattr.seqlim = 100;
    seqattr.seqtyp = ctSEQINC | ctSEQTRM | ctSEQLIM;

    if ((rc = ctCreateSequence(&seqattr))) {
            printf("Error: Failed to create the sequence: %d\n", rc);
    } else {
            printf("Successfully created the sequence.\n");
    }
```

See also

ctCreateSequence (page 168), **ctDeleteSequence** (page 170), **ctOpenSequence** (page 172), **ctCloseSequence** (page 174), **ctGetSequenceAttrs** (page 176), **ctSetSequenceAttrs** (page 178), **ctGetCurrentSequenceValue** (page 180), **ctSetCurrentSequenceValue** (page 182), **ctGetNextSequenceValue** (page 184)

ctDeleteSequence

Declaration

```
NINT ctDeleteSequence(pTEXT seqnam);
```

Description

Deletes the sequence having the specified name from the database.

Return Values

Value	Symbolic Constant	Explanation
0	**NO_ERROR**	Successfully created the sequence.
900	**SEQDUP_ERR**	A sequence having the specified name already exists.
903	**SEQTYP_ERR**	The specified sequence type contains an invalid combination of sequence type options.
904	**SEQINI_ERR**	The initial value specified for the sequence is out of range.
905	**SEQCUR_ERR**	The current value specified for the sequence is out of range.
906	**SEQLIM_ERR**	The limit value specified for the sequence is out of range.
907	**SEQINC_ERR**	The increment value specified for the sequence is out of range.
101	**INOT_ERR**	No sequence exists that has the specified name.

See c-treeACE Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid error values.

Example

```
ctSEQATTR   seqattr;
    NINT            rc;
    /*
    ** Create an incrementing sequence that starts with 1, increments by 3, and
    ** terminates with 100.
    */
    strcpy(seqattr.seqnam, "MyFirstSequence");
    seqattr.seqini = 1;
    seqattr.seqinc = 3;
    seqattr.seqlim = 100;
    seqattr.seqtyp = ctSEQINC | ctSEQTRM | ctSEQLIM;

    if ((rc = ctCreateSequence(&seqattr))) {
            printf("Error: Failed to create the sequence: %d\n", rc);
    } else {
            printf("Successfully created the sequence.\n");
    }
```

See also

ctCreateSequence (page 168), **ctDeleteSequence** (page 170), **ctOpenSequence** (page 172), **ctCloseSequence** (page 174), **ctGetSequenceAttrs** (page 176), **ctSetSequenceAttrs** (page 178), **ctGetCurrentSequenceValue** (page 180), **ctSetCurrentSequenceValue** (page 182), **ctGetNextSequenceValue** (page 184)

ctOpenSequence

Declaration

```
NINT ctOpenSequence(pTEXT seqnam, pLONG pseqhnd);
```

Description

Opens the sequence having the specified name, *seqnam*. *pseqhnd* is a pointer to memory where the sequence handle is set if the sequence is successfully opened.

Return Values

Value	Symbolic Constant	Explanation
0	**NO_ERROR**	Successfully created the sequence.
900	**SEQDUP_ERR**	A sequence having the specified name already exists.
903	**SEQTYP_ERR**	The specified sequence type contains an invalid combination of sequence type options.
904	**SEQINI_ERR**	The initial value specified for the sequence is out of range.
905	**SEQCUR_ERR**	The current value specified for the sequence is out of range.
906	**SEQLIM_ERR**	The limit value specified for the sequence is out of range.
907	**SEQINC_ERR**	The increment value specified for the sequence is out of range.
901	**SEQNAM_ERR**	No sequence exists that has the specified name.

See c-treeACE Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid error values.

Example

```
ctSEQATTR   seqattr;
    NINT            rc;
    /*
    ** Create an incrementing sequence that starts with 1, increments by 3, and
    ** terminates with 100.
    */
    strcpy(seqattr.seqnam, "MyFirstSequence");
    seqattr.seqini = 1;
    seqattr.seqinc = 3;
    seqattr.seqlim = 100;
    seqattr.seqtyp = ctSEQINC | ctSEQTRM | ctSEQLIM;

    if ((rc = ctCreateSequence(&seqattr))) {
            printf("Error: Failed to create the sequence: %d\n", rc);
    } else {
            printf("Successfully created the sequence.\n");
    }
```

See also

ctCreateSequence (page 168), **ctDeleteSequence** (page 170), **ctOpenSequence** (page 172), **ctCloseSequence** (page 174), **ctGetSequenceAttrs** (page 176), **ctSetSequenceAttrs** (page 178), **ctGetCurrentSequenceValue** (page 180), **ctSetCurrentSequenceValue** (page 182), **ctGetNextSequenceValue** (page 184)

ctCloseSequence

Declaration

```
NINT ctCloseSequence(LONG seqhnd);
```

Description

Closes the sequence having the specified handle.

Return Values

Value	Symbolic Constant	Explanation
	NO_ERROR	Successfully created the sequence.
900	**SEQDUP_ERR**	A sequence having the specified name already exists.
903	**SEQTYP_ERR**	The specified sequence type contains an invalid combination of sequence type options.
904	**SEQINI_ERR**	The initial value specified for the sequence is out of range.
905	**SEQCUR_ERR**	The current value specified for the sequence is out of range.
906	**SEQLIM_ERR**	The limit value specified for the sequence is out of range.
907	**SEQINC_ERR**	The increment value specified for the sequence is out of range.
901	**SEQNAM_ERR**	An invalid sequence name was specified: the name is NULL, empty, or too long.

See c-treeACE Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid error values.

Example

```
ctSEQATTR   seqattr;
    NINT            rc;
    /*
    ** Create an incrementing sequence that starts with 1, increments by 3, and
    ** terminates with 100.
    */
    strcpy(seqattr.seqnam, "MyFirstSequence");
    seqattr.seqini = 1;
    seqattr.seqinc = 3;
    seqattr.seqlim = 100;
    seqattr.seqtyp = ctSEQINC | ctSEQTRM | ctSEQLIM;

    if ((rc = ctCreateSequence(&seqattr))) {
            printf("Error: Failed to create the sequence: %d\n", rc);
    } else {
            printf("Successfully created the sequence.\n");
    }
```

See also

ctCreateSequence (page 168), **ctDeleteSequence** (page 170), **ctOpenSequence** (page 172), **ctCloseSequence** (page 174), **ctGetSequenceAttrs** (page 176), **ctSetSequenceAttrs** (page 178), **ctGetCurrentSequenceValue** (page 180), **ctSetCurrentSequenceValue** (page 182), **ctGetNextSequenceValue** (page 184)

ctGetSequenceAttrs

Declaration

```
NINT ctGetSequenceAttrs(LONG seqhnd, pctSEQATTR pseqattr);
```

Description

Retrieves the attributes for the specified sequence. *seqhnd* specifies the sequence handle. *pseqattr* points to a *ctSEQATTR* structure that receives the sequence attribute values.

Return Values

Value	Symbolic Constant	Explanation
0	**NO_ERROR**	Successfully created the sequence.
900	**SEQDUP_ERR**	A sequence having the specified name already exists.
903	**SEQTYP_ERR**	The specified sequence type contains an invalid combination of sequence type options.
904	**SEQINI_ERR**	The initial value specified for the sequence is out of range.
905	**SEQCUR_ERR**	The current value specified for the sequence is out of range.
906	**SEQLIM_ERR**	The limit value specified for the sequence is out of range.
907	**SEQINC_ERR**	The increment value specified for the sequence is out of range.
901	**SEQNAM_ERR**	An invalid sequence name was specified: the name is NULL, empty, or too long.

See c-treeACE Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid error values.

Example

```
ctSEQATTR   seqattr;
    NINT            rc;
    /*
    ** Create an incrementing sequence that starts with 1, increments by 3, and
    ** terminates with 100.
    */
    strcpy(seqattr.seqnam, "MyFirstSequence");
    seqattr.seqini = 1;
    seqattr.seqinc = 3;
    seqattr.seqlim = 100;
    seqattr.seqtyp = ctSEQINC | ctSEQTRM | ctSEQLIM;

    if ((rc = ctCreateSequence(&seqattr))) {
            printf("Error: Failed to create the sequence: %d\n", rc);
    } else {
            printf("Successfully created the sequence.\n");
    }
```

See also

ctCreateSequence (page 168), **ctDeleteSequence** (page 170), **ctOpenSequence** (page 172), **ctCloseSequence** (page 174), **ctGetSequenceAttrs** (page 176), **ctSetSequenceAttrs** (page 178), **ctGetCurrentSequenceValue** (page 180), **ctSetCurrentSequenceValue** (page 182), **ctGetNextSequenceValue** (page 184)

ctSetSequenceAttrs

Declaration

```
NINT ctSetSequenceAttrs(LONG seqhnd, pctSEQATTR pseqattr);
```

Description

Sets the attributes for the specified sequence.

- *seqhnd* specifies the sequence handle.
- *pseqattr* points to a *ctSEQATTR* structure that contains the modified sequence attribute values.

The following attributes can be changed:

- *seqnam* - Changing the sequence name renames the sequence.
- *seqini* - If the sequence enforces a limit, the initial sequence value must be less than the sequence limit for an incrementing sequence or must be greater than the sequence limit for a decrementing sequence.
- *seqcur* - For an incrementing sequence, the current sequence value must be greater than or equal to the initial sequence value, and if the sequence enforces a limit the current sequence value must be less than or equal to the limit value. For a decrementing sequence, the current sequence value must be less than or equal to the initial sequence value, and if the sequence enforces a limit the current sequence value must be greater than or equal to the sequence limit. If *sequnk* is set to a non-zero value, the *seqcur* value is ignored, and the current sequence value is set to the unknown value.
- *seqinc* - The sequence increment must be a positive value that is less than the difference between the initial sequence value and the sequence limit.
- *seqlim* - The sequence limit must be greater than the initial sequence value for an incrementing sequence or must be less than the initial sequence value for a decrementing sequence. The sequence limit is only enforced if the sequence type specifies the *ctSEQLIM* bit.
- *seqtyp* - The sequence type must be set to either *ctSEQINC* or *ctSEQDEC*, and either *ctSEQCYC* or *ctSEQTRM*, and optionally includes *ctSEQLIM*. By changing *seqtyp*, a sequence can be changed from an incrementing to a decrementing sequence and/or from a cycling to a terminating sequence, and a sequence limit can be enabled or disabled. If changing a sequence from incrementing to decrementing (or vice-versa), the initial and limit values must also be changed (see notes on *seqini* and *seqlim*).
- *sequnk* - Setting the sequence unknown flag causes the current sequence value to be set to the unknown value

Return Values

Value	Symbolic Constant	Explanation
0	**NO_ERROR**	Successfully created the sequence.
900	**SEQDUP_ERR**	A sequence having the specified name already exists.

Value	Symbolic Constant	Explanation
903	**SEQTYP_ERR**	The specified sequence type contains an invalid combination of sequence type options.
904	**SEQINI_ERR**	The initial value specified for the sequence is out of range.
905	**SEQCUR_ERR**	The current value specified for the sequence is out of range.
906	**SEQLIM_ERR**	The limit value specified for the sequence is out of range.
907	**SEQINC_ERR**	The increment value specified for the sequence is out of range.
901	**SEQNAM_ERR**	An invalid sequence name was specified: the name is NULL, empty, or too long.

See c-treeACE Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid error values.

Example

```
ctSEQATTR   seqattr;
    NINT            rc;
    /*
    ** Create an incrementing sequence that starts with 1, increments by 3, and
    ** terminates with 100.
    */
    strcpy(seqattr.seqnam, "MyFirstSequence");
    seqattr.seqini = 1;
    seqattr.seqinc = 3;
    seqattr.seqlim = 100;
    seqattr.seqtyp = ctSEQINC | ctSEQTRM | ctSEQLIM;

    if ((rc = ctCreateSequence(&seqattr))) {
            printf("Error: Failed to create the sequence: %d\n", rc);
    } else {
            printf("Successfully created the sequence.\n");
    }
```

See also

ctCreateSequence (page 168), **ctDeleteSequence** (page 170), **ctOpenSequence** (page 172), **ctCloseSequence** (page 174), **ctGetSequenceAttrs** (page 176), **ctSetSequenceAttrs** (page 178), **ctGetCurrentSequenceValue** (page 180), **ctSetCurrentSequenceValue** (page 182), **ctGetNextSequenceValue** (page 184)

ctGetCurrentSequenceValue

Declaration

```
NINT ctGetCurrentSequenceValue(LONG seqhnd, pLONG8 pcurval, pNINT punkval);
```

Description

Reads the current value for the specified sequence. The current value of a sequence can be one of the following:

- The initial value specified when the sequence was created.
- The last value set with the **ctSetCurrentSequenceValue()**, **ctGetNextSequenceValue()**, or **ctSetSequenceAttrs()** functions.
- The unknown value if the sequence has exceeded its minimum or maximum and is not cycling.

Sequence values are stored in the database in which they are defined, and persist between each call to the **ctSetCurrentSequenceValue()** or **ctGetNextSequenceValue()** function.

Return Values

Value	Symbolic Constant	Explanation
0	**NO_ERROR**	Successfully created the sequence.
900	**SEQDUP_ERR**	A sequence having the specified name already exists.
903	**SEQTYP_ERR**	The specified sequence type contains an invalid combination of sequence type options.
904	**SEQINI_ERR**	The initial value specified for the sequence is out of range.
905	**SEQCUR_ERR**	The current value specified for the sequence is out of range.
906	**SEQLIM_ERR**	The limit value specified for the sequence is out of range.
907	**SEQINC_ERR**	The increment value specified for the sequence is out of range.
901	**SEQNAM_ERR**	An invalid sequence name was specified: the name is NULL, empty, or too long.

See c-treeACE Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid error values.

Example

```
ctSEQATTR   seqattr;
    NINT            rc;
    /*
    ** Create an incrementing sequence that starts with 1, increments by 3, and
    ** terminates with 100.
    */
    strcpy(seqattr.seqnam, "MyFirstSequence");
    seqattr.seqini = 1;
    seqattr.seqinc = 3;
    seqattr.seqlim = 100;
    seqattr.seqtyp = ctSEQINC | ctSEQTRM | ctSEQLIM;

    if ((rc = ctCreateSequence(&seqattr))) {
            printf("Error: Failed to create the sequence: %d\n", rc);
    } else {
            printf("Successfully created the sequence.\n");
    }
```

See also

ctCreateSequence (page 168), **ctDeleteSequence** (page 170), **ctOpenSequence** (page 172), **ctCloseSequence** (page 174), **ctGetSequenceAttrs** (page 176), **ctSetSequenceAttrs** (page 178), **ctGetCurrentSequenceValue** (page 180), **ctSetCurrentSequenceValue** (page 182), **ctGetNextSequenceValue** (page 184)

ctSetCurrentSequenceValue

Declaration

```
NINT ctSetCurrentSequenceValue(LONG seqhnd, LONG8 newval);
```

Description

Sets the current value for the specified sequence. If *newval* is outside the boundary set by the initial value (at one end) and the lower limit or upper limit (at the other end) for the sequence, **ctSetCurrentSequenceValue()** returns an error, and the sequence value remains unchanged.

You cannot set a sequence to the unknown value.

Improvements to Sequence API and Active Transactions

In V11.5 and later, FairCom sequence support has been improved as follows:

1. By default, sequence creates and deletes within an active transaction use Immediate Independent Commit Transactions (IICT) to commit immediately. If the sequence attribute *ctSEQTRNDEP* is specified when creating the sequence, the creation and deletion of the sequence is committed only when the transaction commits. This option makes it possible to rely upon a transaction abort to undo the sequence create or delete.
2. The functions that update the sequence, **ctSetSequenceAttrs()**, **ctSetCurrentSequenceValue()**, and **ctGetNextSequenceValue()**, commit their changes immediately if possible. If the changes cannot be immediately committed, rather than failing with error 935 (**IICT_FIL**), the changes commit when the transaction commits. The case where these functions cannot immediately commit is when one of these functions is called in the same transaction that created the sequence, and the sequence is using the *ctSEQTRNDEP* option. In that situation, an IICT cannot be used because the file has been updated in the transaction, and so the sequence record remains locked until the transaction commits or aborts.
3. **ctSetSequenceAttrs()**, **ctSetCurrentSequenceValue()**, and **ctGetNextSequenceValue()** left the sequence record locked until the transaction committed. In V11.5 and later, these functions ensure that the record is unlocked before they return (except for the case mentioned in point 2 above, where a transaction is active and IICT cannot be used).
4. **ctCreateSequence()** and **ctDeleteSequence()** failed with error 588 (**CPND_ERR**) if called within an active transaction and *OPS_DEFER_CLOSE* was not in effect. In V11.5 and later, we temporarily enable *OPS_DEFER_CLOSE* in this situation to avoid the error.

Return Values

Value	Symbolic Constant	Explanation
0	**NO_ERROR**	Successfully created the sequence.
900	**SEQDUP_ERR**	A sequence having the specified name already exists.
903	**SEQTYP_ERR**	The specified sequence type contains an invalid combination of sequence type options.
904	**SEQINI_ERR**	The initial value specified for the sequence is out of range.

Value	Symbolic Constant	Explanation
905	**SEQCUR_ERR**	The current value specified for the sequence is out of range.
906	**SEQLIM_ERR**	The limit value specified for the sequence is out of range.
907	**SEQINC_ERR**	The increment value specified for the sequence is out of range.
901	**SEQNAM_ERR**	Invalid sequence name specified (NULL, empty, or too long).

See c-treeACE Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid error values.

Example

```
ctSEQATTR   seqattr;
    NINT            rc;
    /*
    ** Create an incrementing sequence that starts with 1, increments by 3, and
    ** terminates with 100.
    */
    strcpy(seqattr.seqnam, "MyFirstSequence");
    seqattr.seqini = 1;
    seqattr.seqinc = 3;
    seqattr.seqlim = 100;
    seqattr.seqtyp = ctSEQINC | ctSEQTRM | ctSEQLIM;

    if ((rc = ctCreateSequence(&seqattr))) {
            printf("Error: Failed to create the sequence: %d\n", rc);
    } else {
            printf("Successfully created the sequence.\n");
    }
```

See also

ctCreateSequence (page 168), **ctDeleteSequence** (page 170), **ctOpenSequence** (page 172), **ctCloseSequence** (page 174), **ctGetSequenceAttrs** (page 176), **ctSetSequenceAttrs** (page 178), **ctGetCurrentSequenceValue** (page 180), **ctSetCurrentSequenceValue** (page 182), **ctGetNextSequenceValue** (page 184)

ctGetNextSequenceValue

Declaration

```
NINT ctGetNextSequenceValue(LONG seqhnd, pLONG8 pnxtval, pNINT punkval);
```

Description

Reads the next value for the specified sequence. The next value of a sequence can be one of the following:

- The next value, which is computed from the current sequence value and the attributes of the sequence set in **ctCreateSeqence()** or **ctSetSequenceAttrs()**.
- The unknown value if the sequence has exceeded its minimum or maximum and is not cycling (see "Reaching the End of a Terminating Sequence" below).

Sequence values are stored in the database in which they are defined, and persist between each call to the **ctSetCurrentSequenceValue()** or **ctGetNextSequenceValue()** function.

The value of a sequence set by the **ctGetNextSequenceValue()** function persists in the database until the next call to **ctSetCurrentSequenceValue()** or **ctGetNextSequenceValue()** for the sequence, or until the sequence is deleted from the database.

Reaching the End of a Terminating Sequence

If the sequence is a terminating sequence, and **ctGetNextSequenceValue()** attempts to increment the sequence beyond its upper limit (for positive increments) or decrement the sequence beyond its lower limit (for negative increments), the function returns the unknown value and leaves the current sequence value unchanged.

Once a sequence terminates, **ctGetNextSequenceValue()** continues to return the unknown value for the specified sequence until it is reset to a new value with the **ctSetCurrentSequenceValue()** function, or its definition is changed to a cycling sequence by the **ctSetSequenceAttrs()** function.

After changing the sequence definition to cycle, the first use of **ctGetNextSequenceValue()** for the sequence sets and returns its initial value.

Return Values

Value	Symbolic Constant	Value	Explanation
0	**NO_ERROR**	0	Successfully created the sequence.
900	**SEQDUP_ERR**	900	A sequence having the specified name already exists.
903	**SEQTYP_ERR**	903	The specified sequence type contains an invalid combination of sequence type options.
904	**SEQINI_ERR**	904	The initial value specified for the sequence is out of range.
905	**SEQCUR_ERR**	905	The current value specified for the sequence is out of range.

Value	Symbolic Constant	Value	Explanation
906	**SEQLIM_ERR**	906	The limit value specified for the sequence is out of range.
907	**SEQINC_ERR**	907	The increment value specified for the sequence is out of range.
901	**SEQNAM_ERR**	901	Invalid sequence name specified (NULL, empty, or too long).
902	**SEQHND_ERR**	902	Invalid sequence handle.

See c-treeACE Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid error values.

Example

```
ctSEQATTR   seqattr;
    NINT            rc;
    /*
    ** Create an incrementing sequence that starts with 1, increments by 3, and
    ** terminates with 100.
    */
    strcpy(seqattr.seqnam, "MyFirstSequence");
    seqattr.seqini = 1;
    seqattr.seqinc = 3;
    seqattr.seqlim = 100;
    seqattr.seqtyp = ctSEQINC | ctSEQTRM | ctSEQLIM;

    if ((rc = ctCreateSequence(&seqattr))) {
            printf("Error: Failed to create the sequence: %d\n", rc);
    } else {
            printf("Successfully created the sequence.\n");
    }
```

See also

ctCreateSequence (page 168), **ctDeleteSequence** (page 170), **ctOpenSequence** (page 172), **ctCloseSequence** (page 174), **ctGetSequenceAttrs** (page 176), **ctSetSequenceAttrs** (page 178), **ctGetCurrentSequenceValue** (page 180), **ctSetCurrentSequenceValue** (page 182), **ctGetNextSequenceValue** (page 184)

Sequence Attribute Structure

```
#define MAX_SEQ_NAME_LEN 32 /* maximum length of sequence name */

#define ctSEQINC     0x01    /* incrementing sequence */
#define ctSEQDEC     0x02    /* decrementing sequence */
#define ctSEQCYC     0x04    /* cycling sequence */
#define ctSEQTRM     0x08    /* terminating sequence */
#define ctSEQLIM     0x10    /* sequence has a limit */

typedef struct ctseqattr {
    LONG8    seqini; /* initial sequence value */
    LONG8    seqcur; /* current sequence value */
    LONG8    seqinc; /* sequence increment or decrement amount (always positive) */
    LONG8    seqlim; /* sequence limit */
    TEXT     seqtyp; /* sequence type: incrementing or decrementing and cycling or terminating */
    TEXT     sequnk; /* non-zero indicates current sequence value is unknown */
    TEXT     seqnam[MAX_SEQ_NAME_LEN+1]; /* sequence name */
    TEXT     seqpad;
} ctSEQATTR, ctMEM * pctSEQATTR;
```

A.4. Partitioned File Management API

The FairCom DB API and FairCom DB API .NET interface layers have been enhanced with added functions and methods for administering and managing c-treeACE partitioned files. The .Net methods are:

- **CTTable.PartAdminByKey** (page 187)
- **CTTable.PartAdminByName** (page 188)
- **CTTable.PartAdminByNumber** (page 189)

CTTable.PartAdminByKey

Performs an administrative partition operation on the partition specified by key value.

Declaration

```
COUNT PartAdminByKey( IntPtr key, CTPART_MODE_E partmode );
```

Description

PartAdminByKey() manages the partitions for a table.

- *keyval* [in] - pointer to a partition key value that has been transformed
- *partmode* [in] - partition operation mode. *partmode* is one of of the *CTPART_MODE_E* enums:

PURGE	delete a partition
ADD	add a partition
ARCHIVE	archive a partition
BASE	modify the lower limit partition number value
ACTIVATE	activate an archived partition
STATUS	return the partition status in *partstatus*

Return Values

CTDBRET_NOERROR is returned if no error.

If *CTPART_MODE_E.STATUS* is passed as a partition mode, then one of the following *partstatus* values is returned:

pmSTATUSnone	0	Partition member does not exist.
pmSTATUSexst	1	Partition member is active.
pmSTATUSopnd	2	Partition member is active and currently open.
pmSTATUSarhv	3	Partition member is archived.
pmSTATUSpurg	4	Partition member was purged.
pmSTATUSparc	19	Partition member is pending archive.
pmSTATUSppnd	20	Partition member is pending purge.

See Also

CTTable.PartAdminByName(), **CTTable.PartAdminByNumber()**

CTTable.PartAdminByName

Performs an administrative partition operation on the partition specified by partition name.

Declaration

```
COUNT PartAdminByName( String partname, CTPART_MODE_E partmode );
```

Description

PartAdminByName() manages the partitions for a table.

- *partname* [in] - partition name.
- *partmode* [in] - partition operation mode. *partmode* is one of of the *CTPART_MODE_E* enums:

PURGE	delete a partition
ADD	add a partition
ARCHIVE	archive a partition
BASE	modify the lower limit partition number value
ACTIVATE	activate an archived partition
STATUS	return the partition status in *partstatus*

Return Values

CTDBRET_NOERROR is returned if no error.

If *CTPART_MODE_E.STATUS* is passed as a partition mode, then one of the following *partstatus* values is returned:

pmSTATUSnone	0	Partition member does not exist.
pmSTATUSexst	1	Partition member is active.
pmSTATUSopnd	2	Partition member is active and currently open.
pmSTATUSarhv	3	Partition member is archived.
pmSTATUSpurg	4	Partition member was purged.
pmSTATUSparc	19	Partition member is pending archive.
pmSTATUSppnd	20	Partition member is pending purge.

See Also

CTTable.PartAdminByKey(), **CTTable.PartAdminByNumber()**

CTTable.PartAdminByNumber

Performs an administrative partition operation on the partition specified by partition number.

Declaration

```
COUNT PartAdminByNumber( LONG partno, CTPART_MODE_E partmode );
```

Description

PartAdminByKey() manages the partitions for a table.

- *partno* [in] - partition number
- *partmode* [in] - partition operation mode. *partmode* is one of of the *CTPART_MODE_E* enums:

PURGE	delete a partition
ADD	add a partition
ARCHIVE	archive a partition
BASE	modify the lower limit partition number value
ACTIVATE	activate an archived partition
STATUS	return the partition status in *partstatus*

Return Values

CTDBRET_NOERROR is returned if no error.

If *CTPART_MODE_E.STATUS* is passed as a partition mode, then one of the following *partstatus* values is returned:

pmSTATUSnone	0	Partition member does not exist.
pmSTATUSexst	1	Partition member is active.
pmSTATUSopnd	2	Partition member is active and currently open.
pmSTATUSarhv	3	Partition member is archived.
pmSTATUSpurg	4	Partition member was purged.
pmSTATUSparc	19	Partition member is pending archive.
pmSTATUSppnd	20	Partition member is pending purge.

See Also

CTTable.PartAdminByKey(), **CTTable.PartAdminByName()**

A.5. Data Stacks Filter API

This section describes the data filter stack functions:

ctfiltercbAddFilter

ctfiltercbAddFilter() initializes resources that are used when processing the filter. The function receives the parameters passed to the filter.

Declaration

```
NINT ctfiltercbAddFilter(pTEXT libname, pTEXT funcname, pTEXT params,
     pVOID libhandle, ppVOID pflthandle);
```

Description

- *libname* - the filter callback DLL name that was passed to **SetDataFilter()**.
- *funcname* - the filter callback function name that was passed to **SetDataFilter()**.
- *params* - the function parameter string that was passed to **SetDataFilter()**.
- *libhandle* - the application-defined library handle that was set by **ctfiltercbLoadLib()**.
- *pflthandle* - a pointer to an application-defined filter handle. **ctfiltercbAddFilter()** can allocate memory for use by the filter callback function and can return its address in this parameter.

For example, if the parameter string is:

```
@#mycallback.dll#mycallback_function#my_params
```

then *libname* is “mycallback.dll”, *funcname* is “mycallback_function”, and *params* is “my_params”.

You can use c-tree API functions to open other c-tree files on that FairCom Server and you can read and write the files. All operations are done in the context of the database connection in which you called the record read operation that called the filter function.

Return Values

Value	Symbolic Constant	Explanation
0	CTDBRET_OK	Successful operation.

See c-treeACE Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid c-treeACE error values.

Example

```
/*
    Example Code
*/
```

See also

EvaluateFilter, **LoadFilter**, **ctfiltercbRemoveFilter**, **UnloadFilter**, **SetDataFilter**

EvaluateFilter

Function prototype for the filter callback function.

Declaration

```
NINT EvaluateFilter(pCBDLL pcbdll, FILNO datno, pVOID Recptr,
    pConvMap Schema, VRLEN fixlen, VRLEN datlen);
```

Description

This function is defined in the *ctuserx.c* module used to build a filter callback function.

Where:

- *pcbdll* is a pointer to a CBDLL structure (see definition below) that contains the callback filter state variables.
- *datno* is the data file number by which that connection has opened the data file that is being read. In V12 the file number typedef was formally changed from *COUNT*, a two-byte value to *FILNO*, a four-byte value. Refer to this link for compatibility details. Four Byte File Numbering
- *Recptr* is the record image of the record for which the filter is being evaluated.
- *Schema* is the record schema.
- *fixlen* is the size of the fixed-length portion of the record.
- *datlen* is the length of the available data. For variable length records it may be smaller than the length entire record if the buffer is smaller than the complete record length.

CBDLL Structure

```
typedef struct cbdll_t {
    NINT            cbinst;         /* callback instance count       */
    pTEXT           cbdll;                  /* name of callback DLL          */
    pTEXT           cbfnc;                  /* name of callback function     */
    pTEXT           cbprm;                  /* callback function parameters  */
    pVOID           cbdllhnd;               /* handle to callback DLL        */
    pCBFNC          cbfncptr;               /* pointer to callback function  */
    pCBFNCload      cbload;         /* ctfiltercbLoadLib ptr */
    pCBFNCunload    cbunload;               /* ctfiltercbUnloadLib ptr       */
    pVOID           cbfilterhandle; /* user-defined library handle   */
    TEXT            cbfilnam[MAX_NAME]; /* data file name            */
} CBDLL;
```

SetDataFilter() can register a data record filter callback function that resides in a DLL or shared library. Calling **SetDataFilter()** with a filter expression of the form:

```
@#mycallback.dll#mycallback_function#my_params
```

then *libname* is “mycallback.dll”, *funcname* is “mycallback_function”, and *params* is “my_params”.

You can use c-tree API functions to open other c-tree files on that FairCom Server and you can read and write the files. All operations are done in the context of the database connection in which you called the record read operation that called the filter function.

Building a Filter Callback DLL

A simple way to build a data record filter callback DLL is to use the **mtmake** utility to create a makefile that builds a *CTUSER DLL*. The *CTUSER DLL* exports the functions **EvaluateFilter()**, **LoadFilter()**, and **UnloadFilter()**. The source code for these functions is located in the file *ctuserx.c*.

Additional Callback Functions

In addition to the data record filter callback function, a data record filter callback DLL can optionally implement the following functions. These functions are optional and included for extended capabilities within the callback.

- **LoadFilter()**
- **UnloadFilter()**

Errors logged to *CTSTATUS.FCS* when failing to load a filter callback DLL begin with the text

```
"FILTER_CALLBACK_LOAD: ".
```

A callback DLL that is loaded by a call to **SETFLTR()** is unloaded when the filter is disabled, which is done either by a call to **SETFLTR()** to establish a different filter (or with an empty string to disable the filter), or when the file it closed.

A callback DLL can also be specified in an expression that is passed to **UPDCIDX()**. In that case, the DLL is loaded when the file is physically opened, and the DLL remains loaded until the file is physically closed. Note that if the expression passed to **UPDCIDX()** specifies the name of a filter callback DLL, then every time the data file is opened the DLL must be able to be loaded and the callback function must exist in the DLL. Otherwise the attempt to open the data file will fail with error 867 or 868.

Return Values

Value	Symbolic Constant	Description
0	**NO_ERROR**	No error.
867	**CBKD_ERR**	Failed to load the filter callback library. See *CTSTATUS.FCS* for details.
868	**CBKF_ERR**	Failed to resolve the filter callback function in the filter callback DLL. See *CTSTATUS.FCS* for details.

See c-treeACE Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid c-treeACE error values.

Example

```
/*
    Example Code
*/
```

See also

ctfiltercbAddFilter(), **LoadFilter()**, **ctfiltercbRemoveFilter()**, **UnloadFilter()**, **SetDataFilter()**

LoadFilter

Called when c-treeACE loads a filter callback DLL. This function initializes resources that are used by the filter callback DLL.

Declaration

```
NINT LoadFilter(pTEXT libname, ppVOID plibhandle);
```

Where:

- *libname* is the filter callback DLL name that was passed to **SetDataFilter()**.
- *plibhandle* is a pointer to an application-defined library handle.

Description

This function is defined in the *ctuserx.c* module used to build a filter callback function.

LoadFilter() can allocate memory for use by the filter callback function and can return its address in this parameter.

See also

ctfiltercbAddFilter(), **EvaluateFilter()**, **ctfiltercbRemoveFilter()**, **UnloadFilter()**, **SetDataFilter()**

ctfiltercbRemoveFilter

Called when a data record filter is removed. This function cleans up resources that are used by that filter.

Declaration

```
NINT ctfiltercbRemoveFilter(pTEXT libname, pTEXT funcname, pTEXT params,
     pVOID libhandle, pVOID flthandle);
```

- *libname* - the filter callback DLL name that was passed to **SetDataFilter()**.
- *funcname* - the filter callback function name that was passed to **SetDataFilter()**.
- *params* - the function parameter string that was passed to **SetDataFilter()**.
- *libhandle* - the application-defined library handle that was set by **LoadFilter()**.
- *flthandle* - the application-defined filter handle that was set by **ctfiltercbAddFilter()**.

c-treeACE maintains a list of the names of the callback DLLs it has loaded. A DLL remains loaded until c-treeACE shuts down.

Description

LoadFilter() can allocate memory for use by the filter callback function and can return its address in this parameter.

Return Values

Example

```
/*
    Example Code
*/
```

See also

ctfiltercbAddFilter, **EvaluateFilter**, **LoadFilter**, **UnloadFilter**, **SetDataFilter**

UnloadFilter

UnloadFilter() initializes resources that are used when processing the filter. The function receives the parameters passed to the filter.

Declaration

```
NINT UnloadFilter(pTEXT libname, pVOID libhandle)
```

Description

This function is defined in the *ctuserx.c* module used to build a filter callback function.

- *libname* is the filter callback DLL name that was passed to **SetDataFilter()**.
- *libhandle* is the application-defined library handle that was set by **LoadFilter()**.

Return Values

Value	Symbolic Constant	Explanation
0	CTDBRET_OK	Successful operation.

See c-tree Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid c-tree error values.

Example

```
/*
    Example Code
*/
```

See also

ctfiltercbAddFilter(), **EvaluateFilter()**, **LoadFilter()**, **ctfiltercbRemoveFilter()**, **SetDataFilter()**

A.6. Thread Impersonate API

This section describes the functions used to implement thread impersonation:

ctImpersonateTask

Used to cause a connection to impersonate another connection.

Declaration

```
NINT ctImpersonateTask(NINT mode, NINT taskid);
```

Type

ISAM function called from SQL

Description

The Thread Impersonate feature allows a connection to “impersonate” another existing connection. This is most useful for c-treeACE SQL connections that might wish to do some lower-level ISAM type of activity, such as quickly looking up a record in an uncommitted SQL transaction. The target connection must allow the ability to be impersonated.

Note: This function is not supported by the FairCom Server by default. It requires a custom build of the server available by special request.

mode is one of the following:

- *ctIMPallow* - Allow the specified connection or all connections (if *taskid* is *ctIMPall*) to impersonate this connection.
- *ctIMPdisallow* - Do not allow this connection to be impersonated (which is the default setting for a connection). *taskid* is ignored.
- *ctIMPbegin* - Begin impersonation of the connection whose task ID is *taskid*.
- *ctIMPend* - End impersonation of the connection that is currently being impersonated. *taskid* is ignored.

c-treeACE SQL supports a stored procedure used to easily enable impersonation of its current connection. The procedure is:

```
fc_set_impersonation(taskid)
```

Specify a *taskid* of zero to disable impersonation of the connection (the default). Specify a *taskid* of 1 to enable impersonation of the connection by any other connection. Specify a *taskid* greater than 1 to enable impersonation of the connection only by a connection having that specific task ID.

The *ctSNAPSHOT* (https://docs.faircom.com/doc/*ctreeplus/snapshot.htm*) user snapshot report (op-code = *ctPSSuser*) will return the taskid for the current connection in the `ctgums.slOWNR` structure member. You can
use other options to get the user snapshot output for other users. To get the taskid of the current user at the SQL level, call the `fc_get_taskid()` (https://docs.faircom.com/doc/`sqlops/56371.htm`) stored procedure.

Note: This feature is intended for calling ISAM related functions from within either another ISAM connection, or from a higher-level SQL connection. It is not a trivial task to call SQL-related features from a lower-level connection, and this usage is not recommended.

Return Values

Symbolic Constant	Value	Explanation
NO_ERROR	0	Successfully created the sequence
NSUP_ERR	454	Service not supported.
IMPD_ERR	863	The request to impersonate the specified connection was denied because the target connection does not allow impersonation
IMPU_ERR	864	The request to impersonate the specified connection was denied because the target connection does not allow impersonation by the specified connection.
IMPA_ERR	865	The request to impersonate the specified connection was denied because the target connection is already being impersonated.
IMPB_ERR	866	The request to impersonate the specified connection was denied because the target connection is executing a database operation or is blocked.

See c-treeACE Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid error values.

Example

```
    REGCTREE("inst1");
    if ((rc = INTISAMX(6,           /* index buffers */
                    10,                     /* files */
                    4,                      /* page sectors */
                    6,                      /* data file buffers */
                    usrprf,         /* UserProfile*/
                    uid,                    /* user id */
                    upw,                    /* user password */
                    svn))) {        /* server name */
            ctrt_printf("Error:     Failed to connect to c-tree Server %s instance #1: %d\n", svn, rc);
            goto err_ret;
    }

<< get taskid1 >>

    REGCTREE("inst2");
    if ((rc = INTISAMX(6,           /* index buffers */
                    500,            /* files */
                    4,              /* page sectors */
                    6,              /* data file buffers */
                    usrprf,         /* UserProfile*/
                    uid,            /* user id */
                    upw,            /* user password */
                    svn))) {/* server name */
            ctrt_printf("Error: Failed to connect to c-tree Server %s instance #2: %d\n", svn, rc);
            goto err_ret;
    }

<< get taskid2 >>

SWTCTREE("inst1");
if ((rc = ctImpersonateTask(ctIMPallow, taskid2))) {
            printf("Error: Failed to enable impersonation: %d\n", rc);
            goto err_ret;
    }
printf("Successfully enabled impersonation.\n");

<< open file and add record >>

SWTCTREE("inst2");
if ((rc = ctImpersonateTask(ctIMPbegin, taskid1))) {
            printf("Error: Failed to begin impersonation: %d\n", rc);
            goto err_ret;
    }
printf("Successfully impersonated first connection.\n");

<< open file and read record from taskid1 >>

if ((rc = ctImpersonateTask(ctIMPend, 0))) {
            qa_printf("Error: Failed to stop impersonation: %d\n", rc);
    }
```

ctdbBeginImpersonation

Enables the ability of the thread to support impersonation by another thread.

Declaration

```
CTDBRET ctdbBeginImpersonation(CTHANDLE Handle, NINT taskId)
```

Description

Start the impersonation of the specific task ID.

Stores the current "lock mode" in the *prevlockmode* session property and begins the impersonation for the given *taskId* (**ctImpersonateTask()**).

Retrieves the server-side "lock mode" **ctdbGetLockMode()** after the impersonation and updates the *lockmode* session property.

Note: This function is not supported by the FairCom Server by default. It requires a custom build of the server available by special request.

Returns

CTDBRET_OK on success.

See Also

ctdbEndImpersonation()

ctdbEndImpersonation

Disables the ability of the thread to support impersonation by another thread.

Declaration

```
CTDBRET ctdbEndImpersonation(CTHANDLE Handle)
```

Description

Finishes current impersonation (**ctImpersonateTask()**) and updates the "lock mode" on the server side with the lock mode stored before the impersonation in the *prelockmode* session property.

Note: This function is not supported by the FairCom Server by default. It requires a custom build of the server available by special request.

Returns

CTDBRET_OK on success.

See Also

ctdbBeginImpersonation()

CTSession::BeginImpersonation

Enables the ability of the thread to support impersonation by another thread.

Declaration

```
void CTSession::BeginImpersonation( NINT taskId )
```

Description

Start the impersonation of the specific task ID.

Stores the current "lock mode" in the *prevlockmode* session property and begins the impersonation for the given *taskId* (**ctImpersonateTask()**).

Retrieves the server side "lock mode" **ctdbGetLockMode()** after the impersonation and updates the *lockmode* session property.

Note: This function is not supported by the FairCom Server by default. It requires a custom build of the server available by special request.

Return

void

See Also

CTSession::EndImpersonation()

CTSession::EndImpersonation

Disables the ability of the thread to support impersonation by another thread.

Declaration

```
void CTSession::EndImpersonation()
```

Description

Finishes current impersonation (**ctImpersonateTask()**) and updates the "lock mode" on the server side with the lock mode stored before the impersonation in the *prelockmode* session property.

Note: This function is not supported by the FairCom Server by default. It requires a custom build of the server available by special request.

Return

void

See Also

CTSession::BeginImpersonation()

A.7. Low-Level Functions

This section describes the low-level functions:

c-treeACE Functions to Transfer Files

It is often advantageous for applications to deal with files external to the data and index files. Consider the case of a document. While it may be considered as information related to data record, it is useful to maintain the document external to the data file. To allow a convenient method of moving these files between a client and server, a c-treeACE API function is available to transfer the file from client to server or vice versa.

ctTransferFile() is used to initiate this transfer. The complete API function description is below:

ctTransferFile()

Transfers a file between a c-treeACE client and server. This function works on *non*-c-tree files as well as c-tree files.

Declaration

```
LONG ctTransferFile(pctXFRFIL pxfr);
```

Description

The c-treeACE API function **ctTransferFile()** is used to transfer a file between a c-treeACE client and the c-treeACE database server.

Note:

- The file transfer function allows the calling client process to read the contents of any c-tree or non-c-tree file that the operating system allows the c-tree Server process to open for read access.
- The file transfer function allows the calling client to write to any c-tree or non-c-tree file that the operating system allows the c-tree Server process to open for write access.
- *pxfr* is a pointer to a file transfer request structure, which is defined as follows in c-treeACE V10.3.1 and later:

```
typedef struct ctxfrfil {
 LONG    ver;     /* Version of this structure    */
 LONG    mode;    /* Transfer mode                */
 pTEXT   srcnam;  /* Name of source file          */
 pTEXT   dstnam;  /* Name of destination file     */
 LONG    blksiz;  /* Transfer block size          */
 LONG    ackint;  /* Acknowledgment interval      */
 pXFRFNC xfrfnc;  /* File transfer callback func  */
 pXFRINF xfrinf;  /* File transfer callback data  */
 pVOID   xfrufn;  /* File transfer user callback  */
 pVOID   xfrudt;  /* File transfer user data      */
} ctXFRFIL;
```

For security reasons, this feature is disabled, by default, and requires the calling user to be a member of the ADMIN group. To enable this feature, specify the following in the c-treeACE configuration file:

```
ENABLE_TRANSFER_FILE_API YES
```

A **NSUP_ERR** error code is returned if `ENABLE_TRANSFER_FILE_API YES` is not in *ctsrvr.cfg*.

To allow a non-ADMIN user to call **ctTransferFile()** specify the following configuration keyword:

```
COMPATIBILITY NONADMIN_TRANSFER_FILE_API
```

To use the file transfer function, allocate a *ctXFRFIL* structure and set its fields as follows:

- *ver* - The version structure *ctXFRFIL_VERS_CUR*.

  Note: Future updates to the **ctTransferFile()** function might change this structure. An application must set the version field of the structure to the macro *ctXFRFIL_VERS_CUR*, which c-tree defines to its current version of the file transfer request structure definition.

- *mode* - Set to one of the following:

```
#define ctXFRFIL_SEND   0x00000001  /* send file to server            */
#define ctXFRFIL_RECV   0x00000002  /* receive file from server       */

#define ctXFRFIL_RAW    0x00000004  /* raw file transfer              */
#define ctXFRFIL_CTREE  0x00000008  /* c-tree file transfer           */

#define ctXFRFIL_OVRWRT 0x00000010  /* overwrite existing file        */

#define ctXFRFIL_MASTER 0x00000020  /* perform transfer with master   */

#define ctXFRFIL_CLBK   0x00000040  /* use callback function          */
#define ctXFRFIL_CREDIR 0x00000080  /* create non-existent directories */
#define ctXFRFIL_CPYFIL 0x00000100  /* used for remote file copy      */
```

OR in the *ctXFRFIL_OVRWRT* option if you wish the destination file to be overwritten if it exists.

- *srcnam* - The source file name. When sending a file, this is the name of the file on the client system that the client reads and sends to the server. When receiving a file, this is the name of the file on the server system that the server reads and sends to the client.
- *dstnam* - The destination file name. When sending a file, this is the name of the file on the server system that the server creates and writes with the data received from the client. When receiving a file, this is the name of the file on the client system that the client creates and writes with the data received from the server.
- *blksiz* - The block size, in bytes, to be used when transferring the file. A buffer of the specified block size is allocated by both the client and server processes.
- *ackint* - The acknowledgment interval. A value of 0 or 1 causes the receiving process to send an acknowledgment message to the sending process after each file transfer message it receives. A value greater than 1 causes the receiving process to send an acknowledgment only after receiving the specified number of file transfer messages from the sending process. This option is ignored for shared memory connections.

Important: The file transfer *ctXFRFIL_OVRWRT* mode causes the specified destination file to be overwritten, so if the destination file exists before the file transfer, its original contents are lost (replaced by the transferred data).

Return

Value	Symbolic Constant	Explanation
0	**NO_ERROR**	Successful operation.
454	**NSUP_ERR**	Operation or service not supported. Using an option unavailable to this library.
871	**XFR_SOPN_ERR**	The file transfer operation failed because the source file could not be opened for reading. Check *sysiocod* for the system error code.
872	**XFR_DOPN_ERR**	The file transfer operation failed because the destination file could not be opened for writing. Check *sysiocod* for the system error code.
873	**XFR_READ_ERR**	The file transfer operation failed because the source file could not be read. Check *sysiocod* for the system error code.

Value	Symbolic Constant	Explanation
874	**XFR_WRITE_ERR**	The file transfer operation failed because the destination file could not be written. Check *sysiocod* for the system error code.
875	**XFR_BCON_ERR**	A bound database connection called the file transfer function, but this function is supported for client connections only.
876	**XFR_BSIZ_ERR**	The file transfer operation failed because the caller specified an invalid file transfer block size.
877	**XFR_SFNM_ERR**	The file transfer operation failed because the caller specified a NULL or empty source file name.
878	**XFR_DFNM_ERR**	The file transfer operation failed because the caller specified a NULL or empty destination file name.
879	**XFR_VER_ERR**	The version of the file transfer structure supplied by the caller is not compatible with the c-tree library's structure definition. Check *sysiocod* for the required file transfer structure version.
880	**XFR_DEXS_ERR**	The file transfer operation failed because the destination file exists and the caller did not specify that the destination file is to be overwritten.
881	**XFR_TREP_ERR**	The file transfer operation between a local and master server failed because the server does not support the transactional replication feature.
882	**XFR_TRLC_ERR**	The file transfer operation between a local and master server failed because the server is not configured as a local server. Use the `REPL_MAPPINGS` configuration keyword option to configure the server as a local server.
454		

See c-treeACE Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid c-treeACE error values.

Example

```
pctXFRFL xferfile;
char srcnam[80];
char dstnam[80];

NINT rc = 0;

memset(srcnam, 0, 80);
memset(dstnam, 0, 80);

strcpy(srcnam, "custmast.dat");
strcpy(dstnam, "custmast.dat");

xferfile = (pctXFRFL) malloc (sizeof(ctXFRFL));
if (!xferfile)
    Handle_Error("Bad malloc on xferfile", 0);

xferfile->ver = ctXFRFL_CUR;
xferfile->mode = ctXFRFIL_RECV | ctXFRFIL_OVRWRT;
xferfile->srcnam = srcnam;
xferfile->dstnam = dstnam;
xferfile->blksiz = 32768;
xferfile->ackint = 0;

rc = ctTransferFile(xferfile);
if (rc)
    Handle_Error("File transfer failed.", rc);
```

ctStatusLogWrite

Writes the specified message to the c-treeACE status log file, *CTSTATUS.FCS*.

Declaration

```
COUNT ctStatusLogWrite(pTEXT msg,NINT err)
```

Description

Where

- *msg* is the message to write to the status log
- *err* is the error code to include in the message.

Return

See c-treeACE Error Codes (https://docs.faircom.com/docs/en/UUID-0c047c50-d940-e823-7ced-5bb4da10a558.html) for a complete listing of valid c-treeACE error values.

Example

Writes a message in the following format to *CTSTATUS.FCS*:

```
ctStatusLogWrite("This is a test", 3);


Wed Jun 10 15:10:27 2009
 - User# 00011       This is a test: 3
```

Limitations

Available only in the Server DLL model (server side).

13. Critical Production Updates

This section refers to c-treeACE updates that can directly impact uptime availability for existing production environments. These exceptional issues are specifically noted here as they have potential to affect production reliability.

13.1 Corrective Checkpoint Logic for Buffer/Cache Pages Missing from Update Lists

The c-treeACE Server had been observed to retain an excessive number of transaction logs with a message similar to the following:

```
Tue Sep  1 00:05:55 2009
 - User# 00122 The number of active log files increased to: 28
Tue Sep  1 00:05:55 2009
 - User# 00122 Cache/Buffer Pending Flush
```

This message indicates that a buffer page has not been flushed and remains on the server's internal list of updated buffers. While this can be expected in normal processing under a large transaction load, two unexpected buffer states have been considered as potential candidates for a "false" retention of the transaction log:

- The buffer does not have its update flag set.
- The buffer is not on the list of updated buffers.

c-treeACE now detects and corrects these unexpected buffer states during checkpoint processing.

In the first case, a placeholder transaction number is associated with the buffer.

To address the second case, a test is made. If a page is found off the update list for *CMTPAG_CHECK_LIMIT* consecutive checkpoints a message is logged to *CTSTATUS.FCS* and

the buffer is subsequently placed on the update list for processing during the next checkpoint. *CMTPAG_CHECK_LIMIT* is set to 6 by default.

The following is a list of potential diagnostic messages that may be seen logged to *CTSTATUS.FCS* when one of these events occur and the action taken:

```
Index buffer on commit node list without update flag...
Data cache on commit data list without update flag...
```

The c-treeACE Server found an index buffer or data cache page on the commit list, however, the buffer's update flag is not set. The c-treeACE Server enters zero transaction numbers into the commit list as this cache page has been written to disk.

```
Index buffer on commit node list is not on update list...
Data cache on commit data list is not on update list...
```

The index buffer or data cache page is on the commit node list, but not on the update list. The c-treeACE Server puts the buffer or cache page on the update list so that a subsequent checkpoint flushes that updated buffer or cache page.

13.2 Ensure Proper Mutex Control When Adding Buffers to Commit List

Internal c-treeACE QA testing observed the following messages logged to *CTSTATUS.FCS*:

```
Tue Jul 21 14:04:15 2009
 - User# 00019      WARNING: ct_cmnod adjusted  ct_cmnod:1 list_count:0 buffers:0 aux:0
Tue Jul 21 14:04:15 2009
 - User# 00019      WARNING: ct_cmnod adjusted  ct_cmnod:-1 list_count:0 buffers:0 aux:0
```

The function that adds an index buffer to the commit node list checks if the buffer is on the list before it acquires the mutex that serializes access to the list. As a result, the function might see the buffer on the commit list when it is not actually on the commit list. In this scenario, it might skip adding the buffer to the commit list, however, still increment the count of buffers that reside on the commit list.

To resolve this problem, the logic has been modified to check if the buffer is on the commit list after acquiring the list mutex. This same approach was applied to logic for handling data cache pages.

13.3 Prevent Overlapping Checkpoints

Additional synchronizations have been put into place to prevent two checkpoints from overlapping ensuring complete serialization of the checkpoint process. If the checkpoint was forced from an application level **CTCHKPNT()** call, and an in-process checkpoint is already in progress, the call returns NO_ERROR and the *sysiocod* is set to **OCHK_COD** (-885). This avoids any additional processing overhead as a checkpoint is already in progress.

A server configuration option is available to disable the check on in-progress checkpoints so they can overlap as previously allowed.

```
COMPATIBILITY CHECKPOINT_OVERLAP
```

13.4 Corrected Dynamic Dump File Extensions for Non-TRANPROC Files

A situation was observed in which c-treeACE failed during a dynamic dump and the following message was noted in the *CTSTATUS.FCS* log:

```
CTSTATUS message: getfile: 0xff fill 522 bytes at offset 0:f903f3ax
```

Due to a miscalculated size, the 'FF' fill that this message reports could write past the end of the buffer, corrupting the heap. To resolve this, the fill is now limited to the buffer size.

Note: This miscalculation only applies to non-transaction controlled files being dumped without the !PROTECT option and a non-zero !EXT_SIZE and the total dump size larger than the first segment. For transaction controlled files, it is more likely that a failed system I/O would lead to this crash condition.

13.5 Correct Initialization of c-treeACE SQL Pointers

A c-treeACE SQL crash was reported. A NULL pointer was found to cause the exception in recently added SQL IDENTITY support. While unable to reproduce this issue, it was determined that a failed pointer initialization could lead to this condition. Additional logic was added to ensure all pointers in this module are now properly initialized.

13.6 Corrected c-treeACE SQL References in an ORDER BY Clause

c-treeACE SQL could fail due to references in an ORDER BY clause being incorrectly treated as outer references. This has been corrected in the parsing phase to no longer set these as outer references.

13.7 Corrected c-treeACE SQL Conditional Expression Buffer Length Exception

An unhandled exception occurred when a SQL query read records from a variable-length table using conditions that triggered an internal conditional expression to be used. A record buffer is generally allocated based on the size of a record that was previously read from the file, and this is expected. c-treeACE logic detects any insufficient record buffer size and returns an indication that the record buffer is too small such that the caller can allocate a sufficiently-sized record buffer and re-read the record. However, the conditional expression logic that determines how many

complete fields from the record the record buffer can contain did not detect the case that the final field in the record only partially fits into the record buffer. To correct this, when the starting offset of the last field fits into the record buffer, this logic also checks that the end of the last field also fits into the record buffer avoiding the buffer overrun.

14. Notable Compatibility Changes

This section lists c-treeACE changes that affect compatibility. It is important to review these issues to ensure that your product functions correctly after upgrading to Version 10.

A.8. V10 Client/Server Compatiblity

c-treeACE includes a number of security updates that required substantial changes to the *FAIRCOM.FCS* user and group information files. In addition, the authentication exchange sequence has been extensively modified to enhance network security. As a result, c-tree clients prior to V10 will be unable to connect to c-treeACE V10 servers. Also, V10 clients will not connect to prior servers. The following errors are now returned in these instances.

LSEC_ERR	941	This c-tree Server requires a secure logon that your c-tree client library does not support. Update your c-tree client library.
LSES_ERR	942	This c-tree client uses a secure logon that your c-tree Server does not support. Update your c-tree Server.
LSEV_ERR	959	This c-tree client uses a different secure logon version than your c-treeACE Server. Update your c-tree client.
PWDC_ERR	961	This c-tree Server requires a secure password transmission method that your c-tree client library does not support. Update your c-tree client library.
PWDS_ERR	962	This c-tree client uses a secure password transmission method that your c-treeACE Server does not support. Update your c-tree Server.

c-tree Plus ODBC Compatibility

Note that these V10 client server compatibility changes will impact existing c-tree Plus ODBC drivers (standalone). Please contact FairCom if you distribute these drivers and migrate to V10. Other options may be more applicable.

V9.1 Client/Server Compatibility

V9.2 introduced a modified version of the FAIRCOM.FCS user definitions. As a result, prior client server compatibility can be affected.

If a client prior to c-treeACE V9.2 tries to open the file *FAIRCOM.FCS!USER.dat* with a c-treeACE Server with an updated *FC_USER* structure, the open will fail with error **UVRC_ERR** (859, The client's structure definition for the file *FAIRCOM.FCS!USER.dat* is out of date. Update your client library.)

If a V9.2 or later client with with an updated *FC_USER* structure attempts to open the file *FAIRCOM.FCS!USER.dat* using a c-treeACE Server without the updated structure, the open will fail with error **UVRS_ERR** (860, The server's structure definition for the file *FAIRCOM.FCS!USER.dat* is out of date. Update your c-treeACE Server.)

14.1 Latest Java Version Requirement

c-treeACE SQL requires the Java 1.7 JDK and JRE environment for Stored Procedures, Triggers, and User Defined Functions (UDFs), JDBC, and FairCom DB API Java (c-treeACE SQL V11.0 requires 1.6 or newer). Java is readily available from the *Oracle Java downloads* (*http://www.oracle.com/technetwork/java/javase/downloads/index.html*) website. c-treeACE supports Java on any platform that the Java environment is currently available, including Windows, AIX, Oracle Sun, and Linux.

Check with FairCom for the latest Java compatibility announcements and availability of the latest Java support.

Windows users, see *Using Java 1.7 or Later on Windows*.

14.2 SERVER_DIRECTORY Configuration Option Deprecated

The `SERVER_DIRECTORY` keyword has been deprecated in c-treeACE versions 9.3 and later. `LOCAL_DIRECTORY` is now the preferred keyword to allow the server to store data and files in an alternative location. The `SERVER_DIRECTORY` option did not offer robust handling of relative file paths. Full paths were the only way to ensure proper functionality. For example, with Replication and restoring dynamic dumps, a relative path caused problems in properly locating files.

To avoid potential problems with the use of this option, it has been disabled. When this option is specified in *ctsrvr.cfg*, the c-treeACE Server fails to start and displays the following message:

```
The SERVER_DIRECTORY option is no longer supported.
Use the LOCAL_DIRECTORY option instead.
```

The message is logged to *CTSTATUS.FCS*. On Unix systems it is also written to standard output and on Windows systems it is displayed in a dialog box when the c-treeACE Server is not running as a Windows service.

Use these configuration keywords to relocate transaction logs if this has been the purpose of using this keyword in the past:

```
LOG_EVEN
LOG_ODD
START_EVEN
START_ODD
```

FairCom recommends all server installations to make the switch from `SERVER_DIRECTORY` to `LOCAL_DIRECTORY` for the easiest migration path in the future. Contact your nearest FairCom office should you have any questions or concerns regarding this change.

14.3 Switch to Sbyte for TINYINT Signed Values in ADO.NET Data Provider

The c-treeACE SQL ADO.NET Data Provider was setting *TINYINT* values as *Byte* rather than *SByte* type causing a return of unsigned instead of signed values. This has been switched. ADO.NET applications using *TINYINT* values should consider this change a compatibility issue and substitute usage accordingly.

14.4 Mapping of c-treeACE SQL ADO.NET Data Provider Time Fields

The c-treeACE SQL Time type (c-tree *CT_TIME*) was originally mapped to a .NET *DateTime* type based on the lack of a Time class in the .NET CLR and that a c-treeACE SQL Time value was best represented by a *DateTime* class due to the point-in-time nature of the pair. It was noted that Microsoft mapped ODBC Time types to *TimeSpan* in .NET and SQL Server 2008 introduced a Time type that also mapped to this class. c-treeACE SQL Time is now mapped to a .NET *TimeSpan* type to better align with the Microsoft chosen mapping.

This is a compatibility change to be noted in migrating c-treeACE SQL ADO.NET Provider applications to c-treeACE SQL V9.3.

Note: The .NET *TimeSpan* class allows negative values which are not valid c-treeACE SQL time values. An attempt to update a database Time value based on a negative *TimeSpan* will result in an overflow/underflow error and proper handling of this exception should be taken.

14.5 Conditional Index Expression Changes

The Conditional Index Expression parser underwent an extensive update from V8 to V9 using the FairCom DB API based parser for better integration throughout c-treeACE. Some expressions, though, were found to give different results due to this change. As these have been identified, corrections have been put into place as described in the following sections.

Revised Integer Comparisons in Expression Parser

It was discovered that key values were missing from conditional indexes composed of the following expression:

```
( field > 2148347647 )
```

Integer comparisons such as the following would also result as a comparison to -1:

```
"(unsigned) CT_INT_fld <= (unsigned)0xffffffff"
```

Date and Time types were also previously treated as integers resulting in similar failed comparisons.

Integer types less than four bytes were stored in a four-byte signed variable resulting in unsigned values cast to signed values for comparison purposes. The integer storage type has been changed from *LONG* (4 bytes) to *LONG8* (8 bytes). Expression evaluation should now match previous behavior.

Corrected CVAL_ERR on Partial Record Reads

A conditional expression failed on a partial record read with error **CVAL_ERR** (598, could not evaluate conditional expression) when all fields in the conditional expression were read with c-treeACE V9, although such an expression had previously succeeded. Version 9 introduced a new underlying architecture for the conditional expression logic. The internal **fixDodaOffsets()** function call failed if a partial record buffer was encountered. This function is now allowed to succeed on a partial read, and fails if the expression parser encounters a field that was not read. This was the behavior prior to V9.

NULL Handling in Filter Expressions

Prior to V9, records values with the NULL attribute set are returned even when a filter was in effect. Beginning with V9, these records do not pass this filter condition. V9 now checks if a field is NULL and treats a NULL value as distinct from a value of 0.

14.6 TCtDataSet Fields Property Type Has Changed in c-treeACE VCL

The c-treeACE VCL *Fields* property of the *TCtDataSet* class had overwritten the generic *Fields* property of the *TDataSet* class (the parent of *TCtDataSet*). c-treeACE VCL used a type of *TCtField* while the parent field type was of *TField*. As the *TCtField* type is meant for internal use, it has been renamed *CtField*.

As a result of this change, c-treeACE VCL now has the default generic *Fields* property from *TDataSet* available in *TCtDataSet* and there are two ways to retrieve field information:

- *TCtTable*.*Fields[<field index>]* works exactly as *TCtTable*.*FieldByName(<field name>)*
- With *TCtField*, specific information remains available as *TCtTable*.*CtFields[<field index>]*

14.7 Rollback and Commit Support in c-treeACE ADO.NET Data Provider

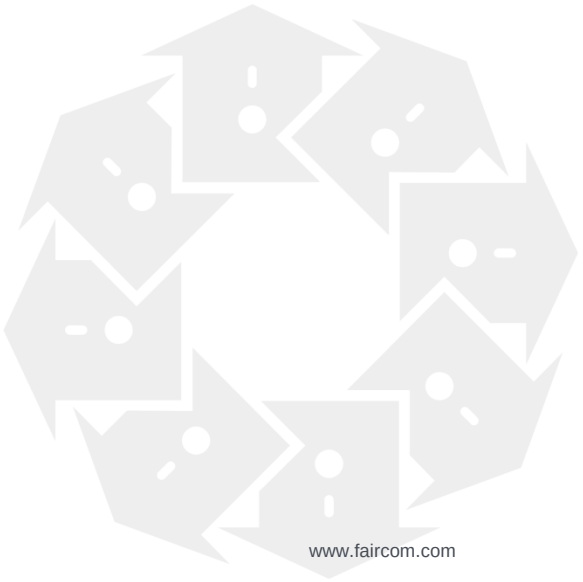
When the ADO provider was first developed, logic was added, with an exception thrown, that prevented a user from calling **Rollback()** and/or **Commit()** while the *DataReader* object was still active (i.e. has not been closed). However, as **Rollback()** and **Commit()** close the *DataReader* rendering it unusable, it was determined this could be problematic as a **Rollback()** may be called by code that does not intend to still use the *DataReader* after the **Rollback()** (and, in any case, a different exception is thrown). This additional check, and the related exception, has been removed.

15. FairCom Typographical Conventions

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

The following formatted items identify special information.

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



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

16. Index

D

E

F

Q

R

S

T

U

V

W