test link

Update Guide

c-treeACE V11.0 Update Guide

Audience

Developers

Subject

c-treeACE V11.0 Update Guide

FairCom®

Contents

1. Introduction

FairCom is pleased to deliver another c-treeACE release. V11 is packed with new features—from minor enhancements to major breakthroughs.

We have implemented many corrections to make our latest release the best ever. For a list of these corrections, be sure to see the V11 Release Notes (https://docs.faircom.com/doc/v11rel/).

2. V11: One Database. Countless Possibilities.

FairCom proudly announces Version 11 of its industry-leading c-treeACE database.

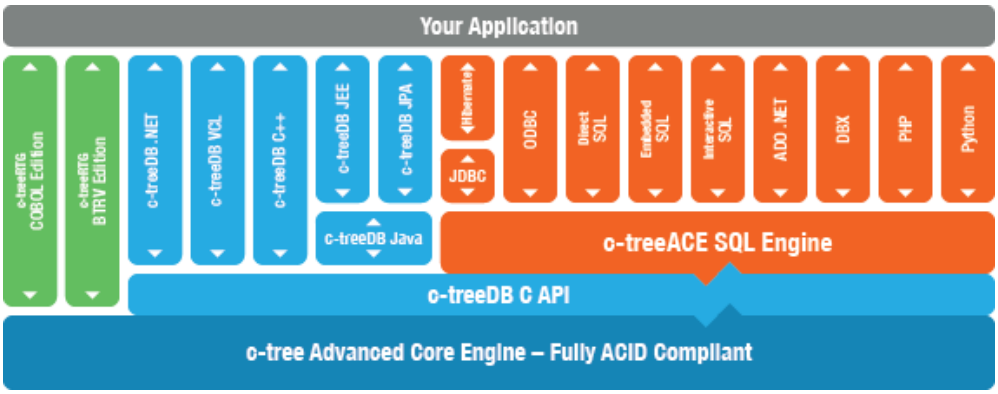
c-treeACE is a modern multimodel database, empowering your application to work directly with non-relational data models and relational SQL simultaneously. c-treeACE includes:

- An ANSI-compliant RDBMS SQL engine with the flexibility to interact with your data and integrate into other systems.
- A fully ACID-compliant data management core (NoSQL) with an advanced low-level indexing infrastructure (i.e., Key-Value Store).

c-treeACE permits NoSQL and SQL access over a single instance of the same data, regardless of your data model. You get incredible performance, control, and reliability of an alternative data model combined with the standards, ease-of-integration, and easy ad hoc query support of SQL. This single database offers countless possibilities.

With a comprehensive set of SQL and NoSQL interfaces providing full read/write access to your data, c-treeACE is the original multimodel database. In fact, over the years we have helped our customers use this advanced multimodel data management infrastructure to implement a number of alternate data models, including: object model; network linked-list model; advanced proprietary models; record-based models; relational models; and many more.

The underlying c-treeACE architecture is a fully ACID-compliant, all-encompassing advanced data store. With many database fundamentals, such as ACID-compliant transactions, row and key level locking, deadlock detection, and more implemented at the NoSQL level (our record/tuple oriented API), you can mix and match any combination of interfaces over a single instance of your alternative data model.



Upgrade to c-treeACE V11

With over 300 enhancements to our previous major release of the c-treeACE database, this guide highlights the top reasons why you should upgrade to c-treeACE V11 today.

2.1 FairCom and Current Database Technology Trends

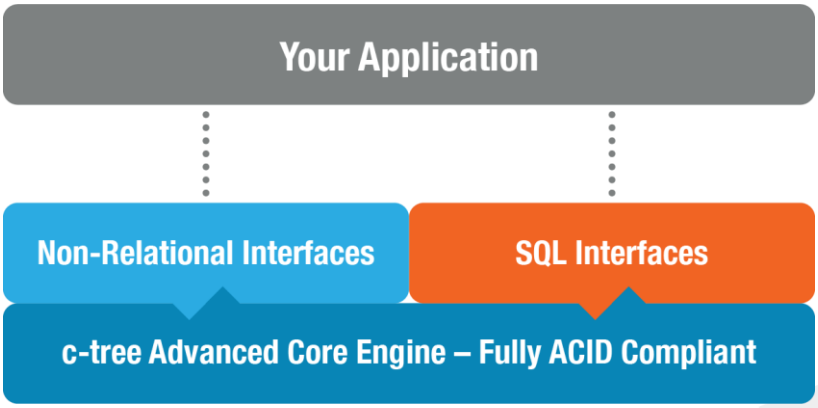
FairCom's c-treeACE database technology enjoys a rich 35-year history as we continue to watch the database industry thrive and change. In fact, at no time in technology history has the database landscape shifted more, and faster, than the last several years. Quickly-changing technologies—and terminologies—bring many new options and questions to data developers and consumers.

A History of Strength

FairCom has always been and remains a champion of solid Online Transaction Processing (OLTP) solutions and flexible data access. We continue to strive to bring you fast, dependable database technology while challenging ourselves to deliver the richest data access toolset on the market. This includes our extensive assortment of ISAM and SQL APIs as well as our unique ability to broaden this scope with implementations of SQL access over non-traditional, unstructured data.

Alternative data models, including record-oriented (ISAM) technology, are categorized today by many as "NoSQL." Newer database technologies have embraced this proven data storage method with many new approaches that mostly ignore a data schema. However, because these newer technologies lack the power of structured query language, they are unable to leverage data to the fullest advantage.

Successful integrated solutions have garnered the term "multimodel database." We believe we are a strong contender in this new multimodel arena—industry analysts such as Gartner are agreeing, as evidenced by the inclusion of FairCom in the 2015 Gartner **Magic Quadrant for Operational Database Management Systems**.



What is NoSQL?

The convergence of cheap storage and massively interconnected social, mobile, and Internet of Things (IoT) data has created a wealth of information to store and search. Not only sheer database size, but the rate of collection has contributed to newer and faster data storage technologies. Merging data from these disparate data sources requires different approaches than traditional relational database products are able to provide.

NoSQL encompasses a wide range of technologies in attempts to solve these data challenges. NoSQL started as "Not Only SQL" implying alternative approaches from structured relational SQL. Each of these technologies targets a specific category of data management:

- **Key-Value Stores** - Fast indexing of unstructured data for instant retrieval without relational management overhead
- **Document Stores** - Typically JSON and XML key-value stores, entire documents can be searched and retrieved
- **Graph Databases** - Interconnected data nodes with relationships. Nodes and relationships contain properties (key-value pairs)
- **Column Stores** - Rapid merging of disparate data sources where data is stored in columns rather than rows as with a traditional relational model

In addition, an assumption of massive scalability and availability is implicit. To accomplish this, compromises must sometimes be made.

Unstructured Data

A key element of today's data storage involves rapidly changing data. Historically, database administrators spent considerable time designing and normalizing data structures for optimal storage size and query performance—"data follows the schema." Today, ever-changing business requirements dictate a need for much more flexible data acquisition. Traditional database schemas constrain the multiple flows of data that can be acquired. Consider merging web server logs with other analytic data. Much of this data fails to follow consistent data schemas. An "unstructured" storage approach is desired. If needed, a structured data query can be defined later—"schema follows the data."

Transactional Technology

Traditional database technology has matured with OLTP applications. Indeed, database benchmarks (TPC-C, TPC-E) are typically a direct measure of OLTP performance. OTLP, strongly rooted in financial and inventory management systems, has classically implied the need for solid transaction integrity, better qualified as ACID—Atomicity, Consistency, Isolation, and Durability. But transaction durability comes with a performance cost. Database vendors have invested heavy intellectual capital in advanced transaction technology for the utmost in performance.

Horizontal scaling of these systems has remained a serious challenge due to the serial nature of transaction processing. Data duplication can be expensive for high-end storage systems and data can become out-of-sync. Eric Brewer's CAP theorem states you can settle for only two out of three states of (C)onsistency, (A)vailability and (P)artition tolerance. Cache coherency issues come into play with multiple servers (partitioning) over a single point of data.

What if the absolute durability constraint can be relaxed? This is the approach taken by many newer database technologies. Allowing "eventual" database consistency provides highly scalable systems, with the premise that searched data, while highly available, may at any times be slightly out-of-date. This is referred to as BASE—Basic Availability, Soft state, Eventual consistency. Most BASE database systems are tuned for write-once, read-often performance. Thus, multiple versions of data may be in the data store at any given time for a period. While acceptable for some applications, this would be highly questionable for financial, or other data that requires an absolute known state at any given point in time.

Where We Are Today

FairCom continues to explore all of these fronts and we find ourselves well positioned in this rapidly changing environment. As we progress, we are confident you will want us for your NoSQL solution:

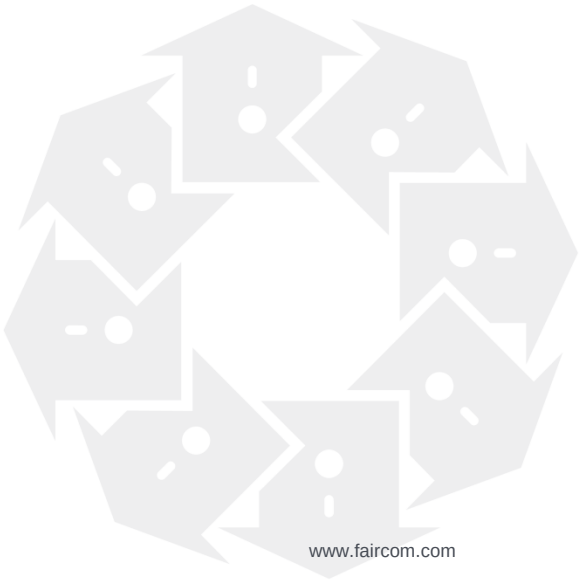
- FairCom has a rock-solid history of database solutions
- “Multimodel Database” is a great description of FairCom’s database technology
- FairCom’s c-treeACE ISAM is a solid Key-Value ACID storage engine—FAST and RECOVERABLE
- FairCom’s c-treeACE handles unstructured data with ease
- FairCom provides the most flexible data access from any interface technology, including SQL access over structured or unstructured NoSQL data

Let FairCom bring value to your data with our No+SQL solutions. Check out these latest V11 No+SQL features that make it possible to implement SQL over NoSQL data:

- FILESET for queries over multiple physical files
- Multi-Schema Table Support: File formats with multiple varying record types mapped virtually into SQL relational-compliant tables.
- SQL Data Arrays for Sub-Record Read Support: Customer-specific propriety formats by which sub-records contain BLOBs/data blocks of embedded variable-length data records.
- Data Types SDK

Do you have a data-related problem you are struggling with? Don’t hesitate to visit with one of our FairCom Engineers and see if we can assist. We’ve helped companies big and small over the years with some very creative and powerful solutions.

Stay Tuned! Our future gets brighter and more exciting every day!



2.2 Great Performance News for OLTP Applications

Performance remains a significant goal of every database system. FairCom continues a tradition of advanced performance solutions balanced with solid data integrity. Our two most powerful OLTP operating models now have additional performance boosting controls:

- A new transaction log mode for Delayed Durability
- New Background Flush controls for safety of non-transaction updates

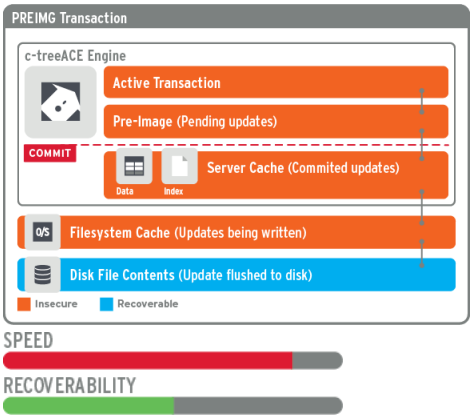
Delayed Transaction Durability

With full transaction control for complete ACID compliance, transaction logs are synced to disk with each commit operation, ensuring absolute data integrity with complete recoverability. Full, durable ACID transaction control enables many powerful features not available without recoverable log data:

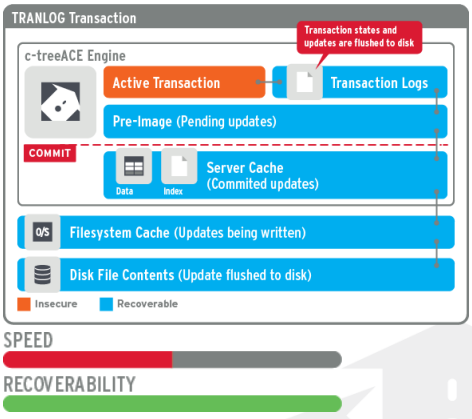
- Automatic database recovery
- Live database backups without rebuild on restore
- Replication
- Transaction auditing

The most critical of these is automatic recovery in case of system failure. Additionally, full transaction control remains a critical area of database performance tuning. Database updates must be secured in write-ahead logs for guaranteed recoverability. This comes with a performance impact due to the synchronous I/O requirements ensuring data is safely persisted.

Preimage (Atomicity Only)



Full Recoverability

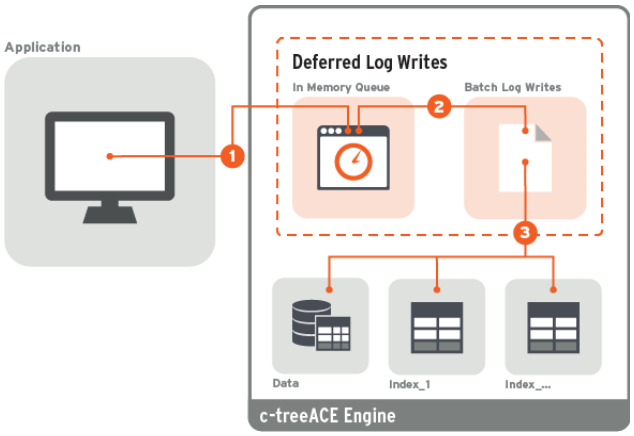


Many applications could benefit from a “relaxed” mode of transaction log writes. With today’s hardware stabilities and power redundancies, it is conceivable to slightly relax full durability constraints and still maintain acceptable risk tolerance. The balance becomes how much loss of recoverability these systems can tolerate.

Allowing database administrators to balance their window of vulnerability against on-line performance, c-treeACE provides a new Delayed Durability (page 25) feature for transaction logs. Regardless of your durability setting, you will always recover to a known state in time, with full referential integrity. The balancing decision becomes how current must you recover? If you are willing to accept guaranteed recovery, say up to one second ago, you can gain a significant amount of performance.

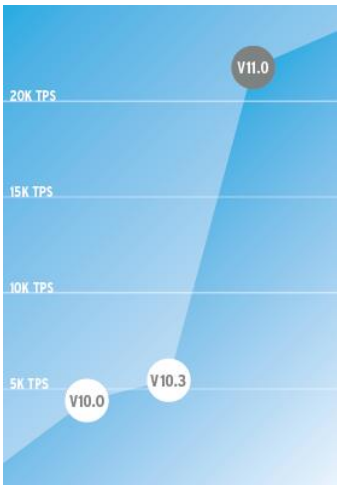
This feature is enabled with the following configuration entry and takes as an argument the maximum number seconds, *N*, that will elapse between log syncs. This ensures that, after a commit, the transaction log will be synced *in no more* than *N* seconds, thereby allowing you to define your window of vulnerability.

```
DELAYED_DURABILITY <N>
```



The end result can approach non-transaction performance while ensuring committed transactions are persisted to storage within less than *N* seconds of vulnerability. In selected test cases, **up to 300% faster transaction throughput** has been observed when configured with 1 second of delayed durability.

Windows Performance



Legend:
V10 - c-treeACE Database Server, Full Transaction Processing, 100 users
V10.3/V1 - c-treeACE / FairCom RTG Database Server, Full Transaction Processing, 100 users
V11.0/V2 - c-treeACE / FairCom RTG Database Server, Full Transaction Processing, 100 users, DELAYED_DURABILITY1

Test Environment:
Dell PowerEdge R710 machine with 2 Xeon 3.6 GHZ Xeon Processors with a total of 32 logical cores, 32 GB RAM, 600G 15000 RPM Drive, Windows Server 2012.

Test Utility:
FairCom's cttctx load test utility simulating a record add / read / delete sequence on 23 files with 4 indexes per file. 100 threads x 10,000 iterations = 1,000,000 transactions per file for a total of 69,000,000 transactions.

Background Flush of PreImage/Non-transaction Data Updates

c-treeACE offers multiple levels of transaction protection for your data. Some applications do not require the recoverability full transaction support provides for performance reasons. However, these applications become quite vulnerable to data loss should system failure occur. If FairCom Server terminates abnormally, say from a power failure, updates to data and index files that are not under full transaction control are lost if those updates have not yet been written from c-tree's in-memory data and index caches to the file system. The following factors typically reduce the number of dirty pages that exist:

1. When an updated cache page is being reused, the updated page is written to the file system cache.
2. When all connections close a c-tree file, FairCom Server writes the updated pages to the file system cache before closing the file.
3. An internal thread periodically checks if FairCom Server is idle, and if so it writes updated pages to the file system cache.

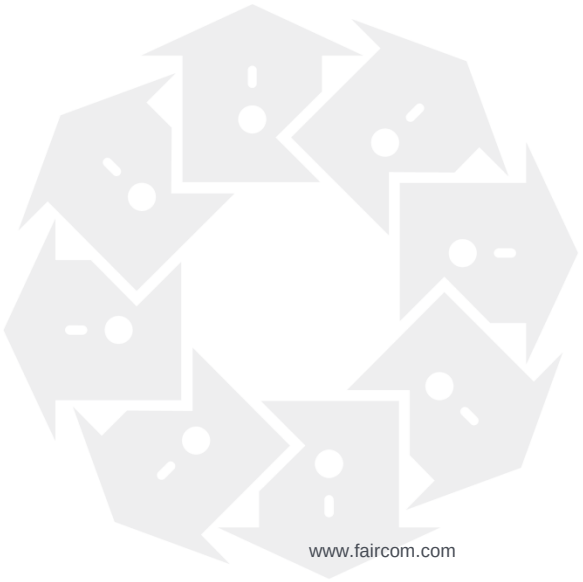
However, the combination of using very large data and index caches, keeping files open for extended periods of time, and constant OLTP activity increase your likelihood that more dirty cache pages exist.

This data vulnerability is now greatly reduced in such a way that allows YOU to define your window of vulnerability. Database administrators can balance performance needs with data safety guarantees to within seconds—or they can select immediate safety.

To set a limit on potential loss of updates for non-transaction data and index files, FairCom Server now supports background flush (page 36) options to write dirty pages to the file system within a specified time period, limiting potential data loss for non-transaction data and index files.

Performance tests have shown that with only a few seconds of data vulnerability, you can obtain the same great performance you are already accustomed to, but now with an assurance that much more of your data is secured to disk thereby substantially reducing your data vulnerability.

See **Delayed Durability Transaction Log Mode for Performance** (page 25).

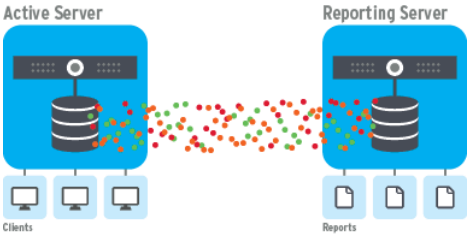


2.3 Replication and Distributed Servers

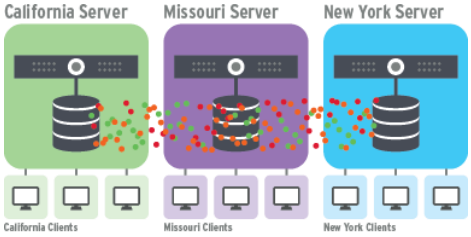
Business challenges, such as high-volume data, dynamic marketing campaigns, and strong competition, have placed demands on IT infrastructure to provide data at an increasingly fast pace. Data must be available anywhere—in real time—to provide information to stay competitive and ahead of the market.

FairCom replication has fast become the solution of choice for distributed database architectures. Real-time database synchronization provides peace of mind when maintaining and synchronizing a physically and logically separate copy of a database. Highly configurable, you decide which data is replicated and where. Typical solutions include:

Backend reporting and data warehousing

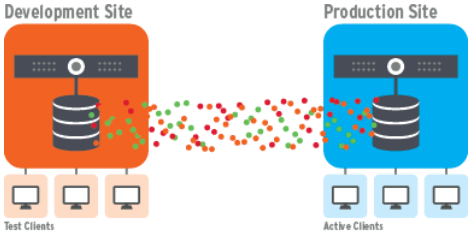


Distributed data for horizontal scaling



Failover solutions for disaster recovery and high availability

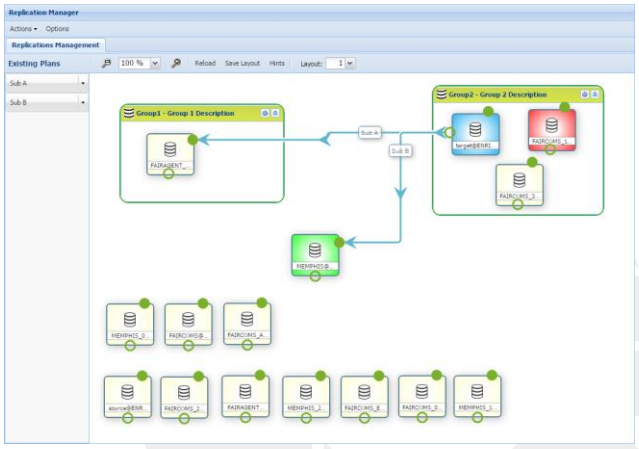
Replica environments for testing and validation



Replication Solutions

FairCom Replication Manager (page 43)

The Replication Manager gives you full control over your entire replication solution, easily handling complex topologies. A web-based view allows complete management of your total environment from within your browser. Templates simplify configuration. Point and click to fine-tune and manage your replication environment.

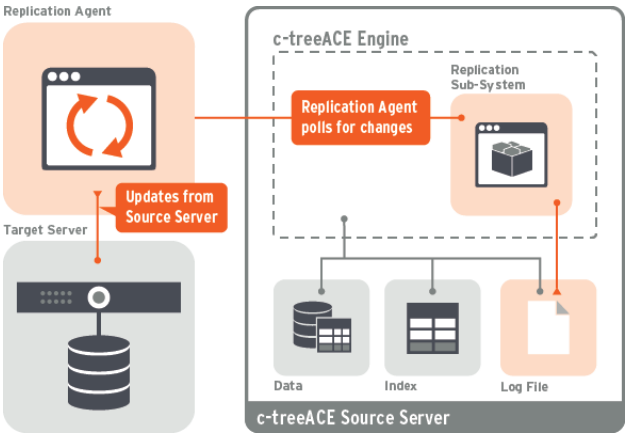


Replication Agent 2.0

The Replication Agent introduces a new, multi-threaded design:

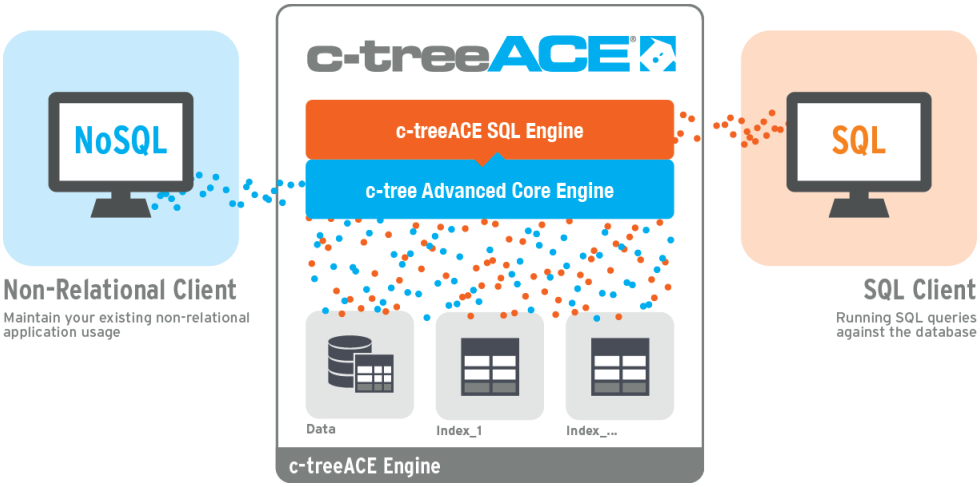
- Previously each connection needed 1 or 2 agents—now a single agent can service multiple c-treeACE connections, reducing overhead and simplifying deployment and administration.

- Sets of files can be assigned to unique Replication Agents. Deploy multiple agents for greater parallel data throughput.
- Event Notification for external agent event handling.
- Ability to start at current source server log position makes it easy when re-synchronizing data.



2.4 No+SQL Integration Enhancements

For over 15 years, FairCom has blended SQL and NoSQL into an integrated database. With over 20 interface technologies providing full read/write access to your data, c-treeACE is the original multimodel database. V11 continues this advanced integration of SQL over NoSQL data with our unique No+SQL solutions.



No+SQL Highlights

Integrating SQL and NoSQL into a single database, c-treeACE delivers new capabilities that are not possible with any other database. With V11, we are expanding our full SQL capabilities over your NoSQL data with features such as the following:

- Multi-Schema Table™ support allows a single table with multiple record types to appear to SQL as multiple virtual tables, each with a single schema. This flexible combination of NoSQL data with SQL APIs is very powerful for blending the performance and control of NoSQL with the standards and ease of integration afforded by SQL.
- The FairCom exclusive FILESET (page 52) feature allows for efficiently querying hundreds or thousands of physical files, eliminating the overhead of a SQL View.
- SQL Data Arrays (page 54) unlock sub-records in scenarios where records may contain multi-schema data. An exclusive c-tree algorithm creates a virtual table for each sub-record type, allowing a SQL query to access this traditionally NoSQL data. Note: This technology is tightly coupled with, and dependent on, our customer's proprietary data formats and requires close communication between our team and yours. Call your nearest FairCom office to discuss your proprietary data challenges.

2.5 Latest c-treeACE SQL Features

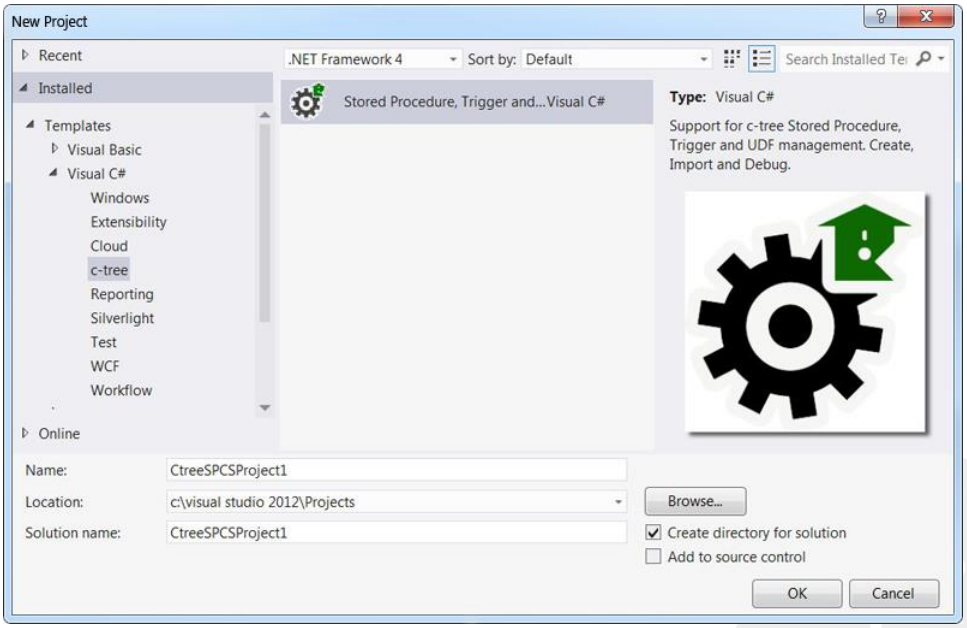
c-treeACE SQL brings standards-compliant SQL to even your most non-SQL data. You've invested years in your successful c-tree applications. Now you can enjoy fresh new interfaces and capabilities while maintaining strong and proven c-tree foundations. Many applications are already reaping the benefits. And with that, many requests for extended SQL capabilities have been suggested. c-treeACE SQL V11 brings a list of great new features.

Stored Procedure Development in the .NET Framework - C#

SQL stored procedures and triggers are an easy and powerful way to move advanced logic into your server core, protecting investments in performance-sensitive processing while maintaining a centrally managed approach for long-term maintenance. Stored procedures fully encapsulate business logic for business rule enforcement. Triggers maintain database integrity directly at the database server level.

c-treeACE SQL has long supported stored procedures, triggers, and user defined functions with Java for cross-platform development ease.

Stored procedures are now available for your Microsoft Windows .NET development environment. Program your procedure in C# or any other .NET supported language. Seamlessly remain in your .NET environment while developing advanced server-side processing logic. With Visual Studio integration, it is easy to create, debug, and deploy your C# stored procedure code remotely. Your assemblies are deployed alongside c-treeACE SQL with a click of a mouse.



See **New Stored Procedure Development Frameworks** (page 58).

NetBeans Plugin for Java Stored Procedure Development

c-treeACE SQL now includes a Java NetBeans plugin (, /doc/jspt/62755.htm) for advanced development potential. Take advantage of the complete NetBeans IDE in creating and maintaining new and existing Java stored procedures. Existing Java stored procedure investments can continue to be used with much easier maintenance and deployment capabilities. See **NetBeans Plugin for Java**.

More SQL Enhancements

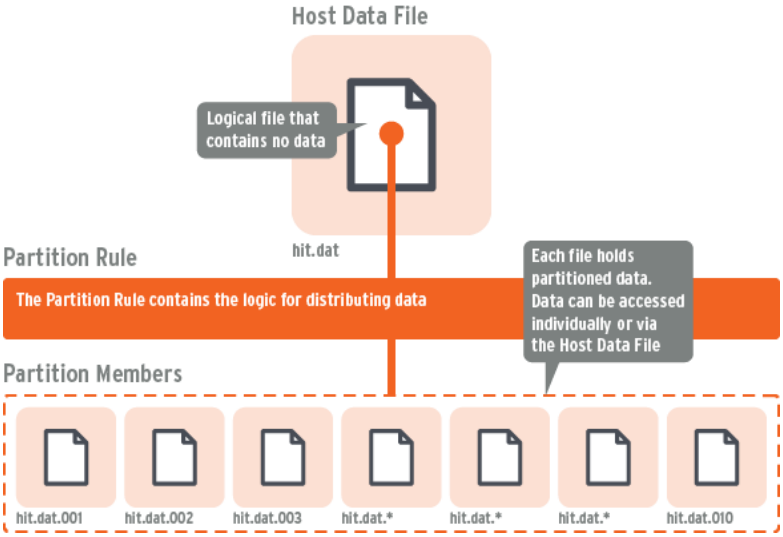
c-treeACE SQL includes a list of many other big new additions for easier and enhanced SQL capabilities with your existing applications:

- **Entity Framework 6** (page 59) Support with ADO.NET
- **SQL Groups** (page 58) for much easier user role management
- **Table Valued Functions** (page 60) for dynamic calculated tables in queries based on stored procedure logic
- **Extended ALTER VIEW, ALTER TABLE, and ALTER INDEX Flexibility** (page 60) for views, tables, and index management
- **Table Locks** (page 64) for exclusive user access
- **Recursive Query** (page 61) support for hierarchal “tree”-based queries
- **Scrollable Cursors** (page 65)
- And continued query optimizer improvements for ever faster performance

Collectively, these great new SQL features bring enhanced administrative capabilities to new **and** existing applications.

2.6 Added Conditional Expression Support for Partitioned Files

c-treeACE has supported advanced partitioned files for many years. A partitioned file logically appears to be one data file (and its associated index files). It is actually a set of files whose contents are physically partitioned by the value of a *partition key* that can be based on a rule or expression. By dynamically splitting data over multiple physical entities, it is much easier to quickly purge or archive volumes of data.



With growing data volumes, this feature is becoming one of the most interesting scaling choices for database performance. One of the past limitations was a partition “rule” for keys was hard coded into our server based on particular needs, typically by date range.

c-treeACE has also long supported conditional expressions for advanced conditional indexing. By bringing our powerful built-in expression parser to partition rules, we’ve extended conditional expressions to the creation of partitioned files. Now it’s easy to create partitioned files at will by defining your partition rule as a conditional expression when creating the file.

An extended set of predefined functions allows extremely flexible rules. For example, partition by day of week, day of month, monthly, quarterly, or yearly. Partitioned Files are now a complete and easy ready-to-use feature for any deployment.

Example

To create a table partitioned by day of month (1-31):

```
CREATE  TABLE  log_data  (timepoint TIMESTAMP, value VARCHAR(256));
CREATE  INDEX  date_partition (timepoint) STORAGE_ATTRIBUTES 'partition=DAYOFMONTH(timestamp)';
```

It’s now that easy! (page 86)

2.7 Java Family Expands

FairCom continues its dedication to cross-platform support and Java remains one of the strongest global standards in this regard. Java retains its position as the language of choice for distributed enterprise software development. V11 expands c-treeACE Java offerings (page 90), with multiple Java technologies for desktop to distributed enterprise solutions:

- c-treeACE JDBC - Type IV JDBC driver for industry-standard applications
- FairCom DB API JEE Enterprise Java Bean - Integrates c-treeACE as a Resource Adapter
- FairCom DB API Java - Fast NoSQL access to c-tree files
- FairCom DB API JPA Java Persistence API - NoSQL Java Persistence API manages data without the need to use SQL

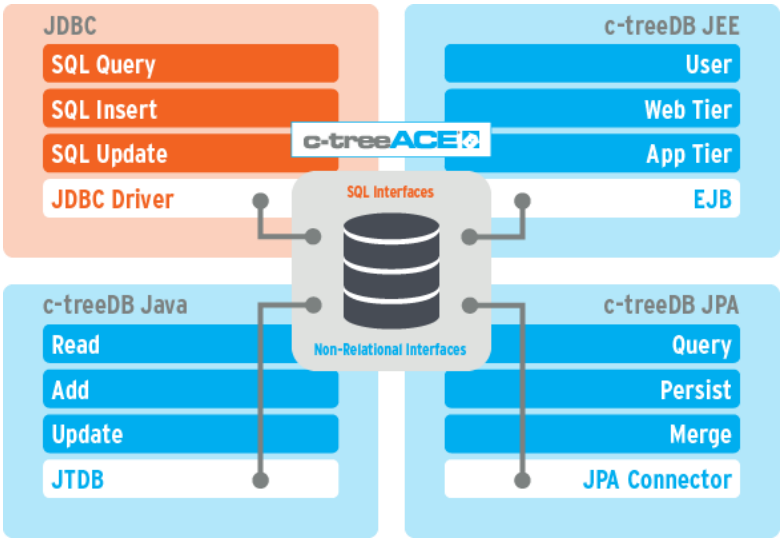
c-treeACE includes everything Java programmers require:

c-treeACE JDBC (page 94): c-treeACE SQL JDBC provides SQL access to c-tree files. The Java Database Connectivity (JDBC) API, the industry standard for database connectivity, provides a standard connection between Java and the c-treeACE SQL database engine. c-treeACE SQL JDBC provides an API that allows you to exploit Java's "Write Once, Run Anywhere" capabilities for applications that run on different platforms and access enterprise data. Use this JDBC support as part of your JPA Hibernate stack for SQL-oriented access as opposed to our unique record-oriented JPA described later.

FairCom DB API JEE (page 94): The FairCom DB API JEE (Java Enterprise Edition) makes NoSQL methods available for enterprise-class, distributed applications, integrating multiple environments, such as mainframes and open systems, that require high-performance NoSQL access to c-tree data. FairCom JEE offers an implementation of an Enterprise Java Bean (EJB) Resource Adapter for JEE Application Servers such as Glassfish. It publishes c-treeACE NoSQL methods, which allows enterprise-class, distributed applications to view c-treeACE data as a registered resource.

FairCom DB API Java (page 94): FairCom DB API Java makes navigational access methods to c-tree files available for Java applications that need to access c-tree data with high throughput, performance, and low-level control. This API (commonly referred to as "JTDB" for short) provides Java developers a unique record-oriented, NoSQL Java framework to manage data with the c-treeACE database engine. This interface offers the performance advantages of direct access to records while still allowing full Java access to the same data available through the industry-standard JDBC interface. FairCom DB API Java is perfect for single-tier Java applications.

FairCom DB API JPA (page 92): The FairCom FairCom DB API JPA (Java Persistence API) provides an object-oriented abstraction layer to manage data without the need to use SQL commands, which allows applications to be independent of any specific database. Unlike many SQL-based JPA implementations, such as Hibernate, FairCom uses our low-overhead, navigational key-value interface technology for improved performance.



And don’t forget, c-treeACE SQL supports development of stored procedures, triggers, and user-defined functions written in Java. Moving your business logic server-side not only improves performance, it also enforces business rules and allows for controlled updates as rules are maintained over time. V11 brings a new NetBeans IDE environment (, /doc/jspt/62755.htm) to your Java stored procedure development platform.

2.8 Advanced Full-Text Search Indexing

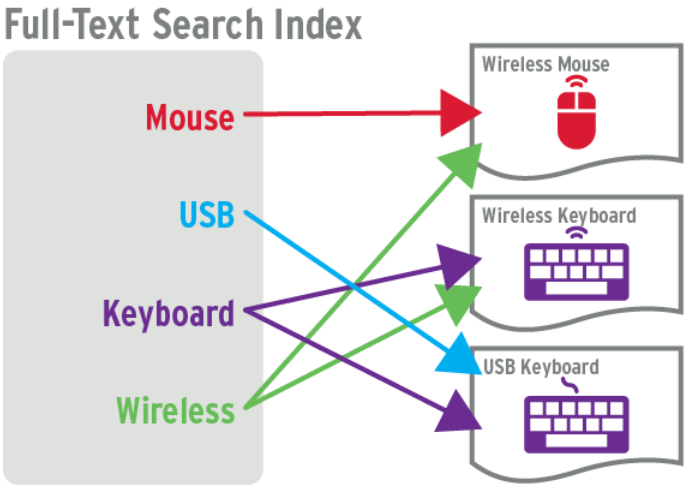
This powerful new index technology is currently under development to support our Record Oriented APIs (CTDB) with plans to extend it further to additional APIs, including SQL in the future.

Similar to a traditional b-tree index over a c-tree data file, you may now define a full-text index by defining which character-type fields to include in this search index. An additional set of full-text index files will be maintained on disk.

This initial support is on a file-by-file basis (same as a typical c-tree index) and we intend on expanding the developer's control over these indexes to include: multi-file-wide; database-wide; or global server-wide support. Once a full-text search index is defined for a file, it is maintained in real-time along with any other b-tree type c-tree indexes.

Accessing data records through a full-text search index is simple. Using new FairCom DB API API search functions, you provide a "word" or "phrase" (text) to be searched. All records whose fields (defined in the index) contain this text are returned. It is left up to the developer to utilize this result as needed for the application.

Today's applications process vast amounts of textual data. Full-Text search is a great mechanism for fast efficient access to character-type data elements. c-tree applications with large volumes of text data can now be complemented with high performing text search capabilities.



2.9 Deferred File and Index Maintenance

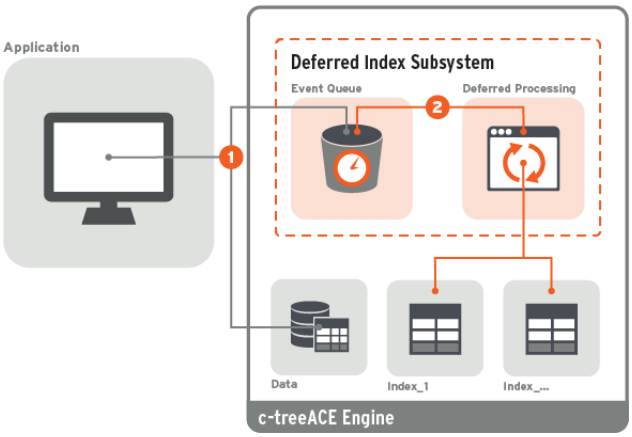
Deferred file and index maintenance postpones certain index and data maintenance operations to background processes, thus offloading performance bottlenecks in various use cases. For example, index update operations can be deferred.

Several very exciting new performance-based indexing features are available with respect to deferred data and index handling.

- A new deferred indexing (page 96) feature allows background processing of new index entries for much faster application response times. For data files with many non-critical indexes, this can bring a big boost in performance when large numbers of new keys are added. Bulk load processes can benefit as essential, primary indexes are immediately available, while non-critical indexes are eventually brought to full consistency.

  In addition, new deferred key loading modes are available for permanent index creation allowing deferred loading of key values, such that your index is immediately available, although in an “eventually consistent” state until all keys are completely loaded. At that time, the index switches into a fully available and consistent mode, unless also marked with the above permanent deferred mode.
- Record Update Callback Functions (page 101, /doc/ctreeplus/record-update-callback.htm) give essential control to call detailed user-defined actions when a record is modified. Much as a notification event, these events handle any record-update operations and are deferred to a background process for handling. For example, an external indexing process can be called to process data in parallel for non-typical data types.

This precise control remains the hallmark of FairCom’s dedication to the database development community.



2.10 Coordinate Application Recovery with Transaction Restore Points

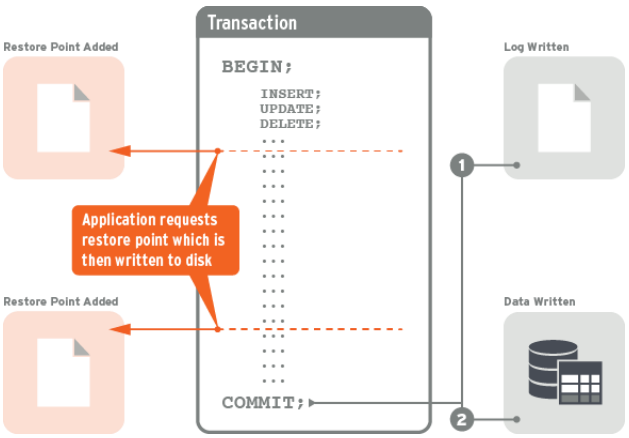
Transaction control by definition requires logging all database changes to persisted storage. Of course, transaction control is highly desirable for recovery of the database in the event of system failure. It also makes available other valuable features such as replication. However, this comes with a direct and measurable performance cost.

The new Delayed Durability feature can greatly enhance performance; however, should a system fail, database recovery could take considerable time if the transaction logs held substantial volumes of data not yet flushed to data and index files.

This leads to an additional challenge of how to balance data integrity using transactions against performance **and** recovery. Certain classes of applications can restore to known points in time. Consider the simple case of a bulk load. Should anything fail during loading, the load process can usually be restarted. If this known position could be coordinated with a c-treeACE server transaction log position, one could architect a very effective recovery process.

To this end, a new Restore Point feature has been introduced. A Restore Point is a position in a transaction log to which FairCom Server's automatic recovery can roll back the system in event of system failure.

The Restore Point is used for restoring a system to a point in time coordinated with the application as a known good state. After restoring the system to a Restore Point, the application can be architected to perform work that was not recovered. For example, those applications maintaining their own journal of operations. By coordinating a known good state in time between the application and the server's transaction logs, these applications can quickly recover back to a point in time when absolutely needed.



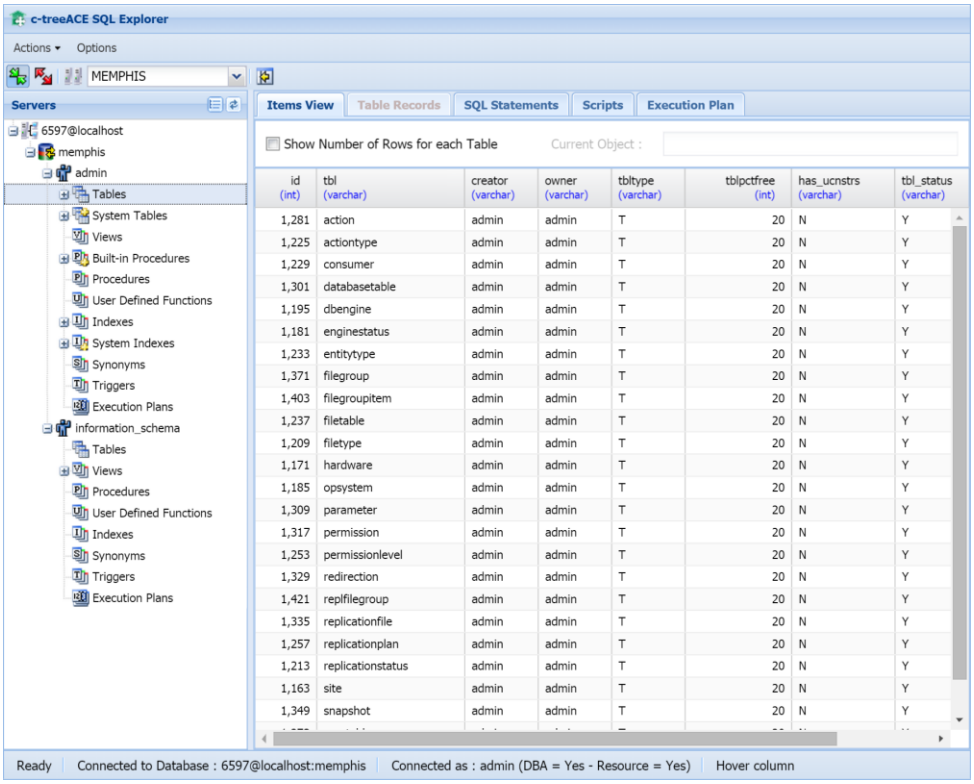
See **Transaction Restore Points for Application Recovery**.

2.11 Browser-Based Administration Tools

Consistent with our new latest releases, we will offer a fresh, familiar interface for interacting with c-treeACE. With a new embedded web service within the c-treeACE Server, administrators can connect directly to a c-treeACE server web port using existing browsers on your desktop. Two web-based features mark their debut in this release:

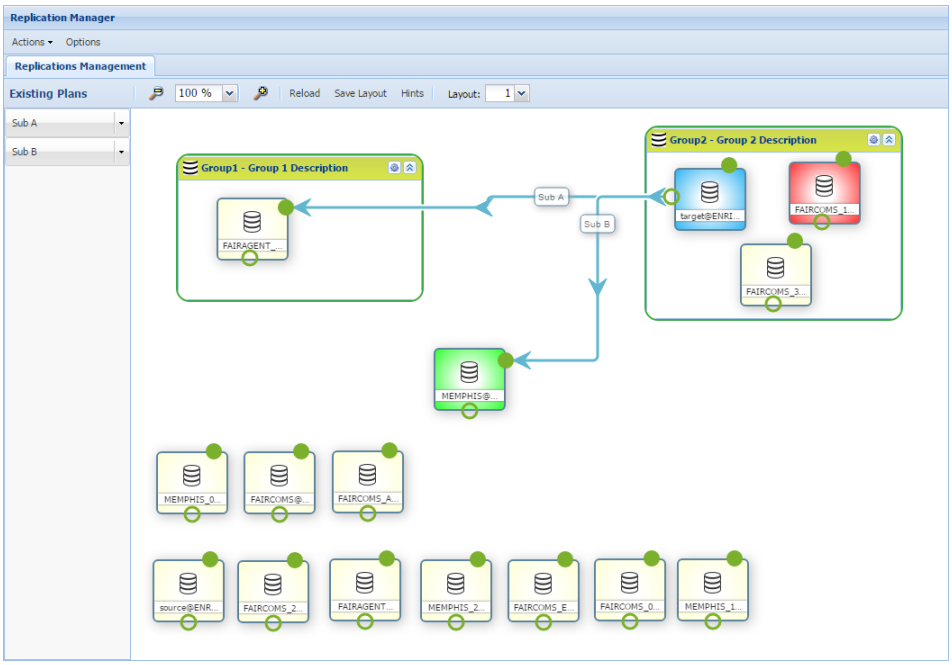
Browser-Based c-treeACE SQL Explorer (page 121)

This is a browser-based version of our popular graphical administration tools. Further extending administrator flexibility, this web-based c-treeACE SQL Explorer can easily be transitioned into any existing web server with c-treeACE SQL PHP support.



Replication Manager (page 43)

This new web-based replication management tool allows sophisticated setup and administration of your multi-server environments with the powerful replication abilities of FairCom Replication Agent and c-treeACE.



c-treeACE SQL PHP, the predominant language for data-driven web applications, also receives updates in V11 as this popular interface continues to grow in popularity.

* Replication Manager is planned for formal release in early 2016. Contact your nearest FairCom office for availability.

2.12 Many Other Requested Features

c-treeACE V11 includes many features requested by our development community. Many of these advanced features go unnoticed—and underutilized. Take a look and you'll likely find a feature or two ready for your immediate use.

Millisecond Timestamp Resolution

We now introduce millisecond timestamp resolution for c-tree time types. This support is available across multiple c-tree APIs, including core c-tree support, FairCom DB API and c-treeACE SQL.

Table Locks

Table Locks prevent other connections from reading and/or writing to the file. These can greatly reduce resource usage and contention with other connected users.

TCP IPv6 Support

Internet Protocol (IP) IPv6 format is available for c-treeACE servers on Windows and Linux (Java and ADO.NET SQL clients currently support only IPv4). IPv6 greatly expands the number of available IP addresses over the previous IPv4 standard.

User-Defined Functions in Conditional Expressions

It is now possible to add a "User-Defined Function" (UDF) to c-tree's expression evaluator, providing UDF support for conditional expressions. A new function pointer type enables this support.

Memory Files

True in-memory files for extremely fast data access performance.

Quiesce and File Block

Block physical access to files while not stopping your server process.

- **Quiesce** is designed for quick SAN snaps and other times when a complete quiet server state is required. Multiple options are available for full or crash consistent data states.
- **File Block** is the perfect solution when you simply need to replace or move a file on an active server. Block the file, replace and unblock. c-treeACE coordinates all client interactions.

Hot Backups

Our Dynamic Dump backup is a real-time hot backup solution. Completely transparent and automatic backups can take place while users stay connected and working.

Full featured restore includes roll forward for the ultimate in data safety.

Notification

Receive file-change notification events. Build a publish/subscribe system to pass record changes around. A notification callback is available to immediately handle change events. Dynamically display notification events directly from your application.

Compare that to the new Record Update Notifications (page 101, /doc/ctreeplus/record-update-callback.htm) which queue and persist changes to a background thread.

Choose the model that best fits your application needs.

Message Queues

General purpose message queues are the backbone of file notification and available for any general messaging use.

Add inter-client communication to your application suite. Build a job queue. Process external events. c-treeACE message queues can be extended to nearly limitless types of message data.

Conditional Indexes

Create high-performance search indexes by conditionally indexing only required data.

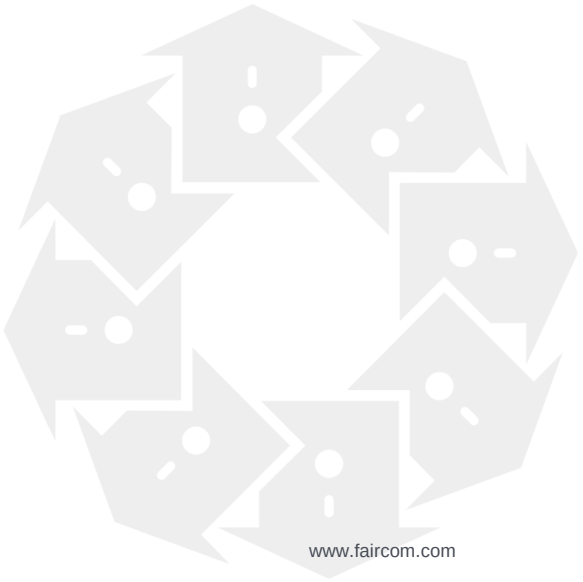
Copy Files Between c-treeACE Servers

Easily moving and copying files between servers is a consistent need we've heard from our customers. A new file copy function makes this a seamless effort directly from the application layer. File transfer is now done server-to-server in addition to the client-server transfer feature introduced in V10.

Don't see something you need? Reach out to your nearest FairCom office and ask our engineering team! We may have related technology already available, and we're always open to fresh suggestions.

3. V11 Performance Gains

FairCom's continued performance improvements speed your application ahead of your competition. These gains are especially apparent when scaling across multiple users and leveraging NoSQL and Big Data.



3.1 Delayed Durability Transaction Log Mode for Performance

Delayed Durability Transaction Processing

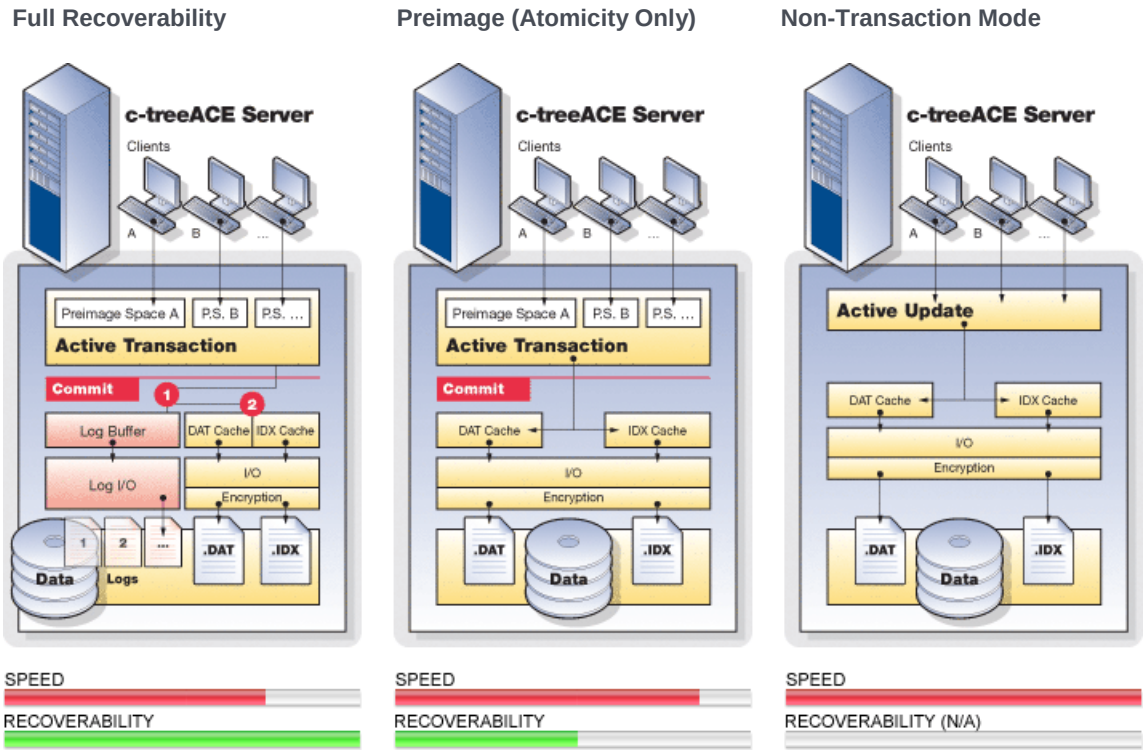
With full transaction control for complete ACID compliance, transaction logs are synced to disk with each commit operation, ensuring absolute data integrity with complete recoverability. Full durable ACID transaction control enables many powerful features not available without the availability of recoverable log data:

- Automatic database recovery
- Live database backups without rebuild on restore
- Replication
- Transaction auditing

The most critical of these is automatic recovery in case of system failure. However, full transaction control remains a critical area of database performance tuning. Database updates must be secured in write-ahead logs for guaranteed recoverability. This comes with a performance impact due to the synchronous I/O requirements ensuring data is safely persisted.

Many applications could benefit from a "relaxed" mode of transaction log writes. With today's hardware and power redundancies, it is conceivable to slightly relax full durability constraints and still maintain an acceptable data risk tolerance. The balance becomes "how much loss of recoverability can these systems tolerate?"

Allowing database administrators to balance their window of vulnerability against online performance, c-treeACE provides a new Delayed Durability feature for transaction logs.



This new transaction operation mode allows its c-treeACE transaction log updates to remain cached in its in-memory transaction log buffer as well as in file system cache, even after a transaction has committed. The challenge is to avoid index and disk updates from reaching disk before the transaction entries do. FairCom has managed to delay transaction log writes to persisted storage while guaranteeing these transaction log entries for a given transaction write to disk before any data file updates associated with that transaction are written to file system cache or to persistent storage with a modified log sync strategy. In addition, a background thread guarantees an upper bound on the total amount of time any transaction remains in a non-synced state.

This feature is enabled with the following configuration entry and takes as an argument the maximum number seconds, *N*, that will elapse between log syncs. This ensures that, after a commit, the transaction log will be synced *in no more* than *N* seconds, thereby allowing you to define your window of vulnerability risk.

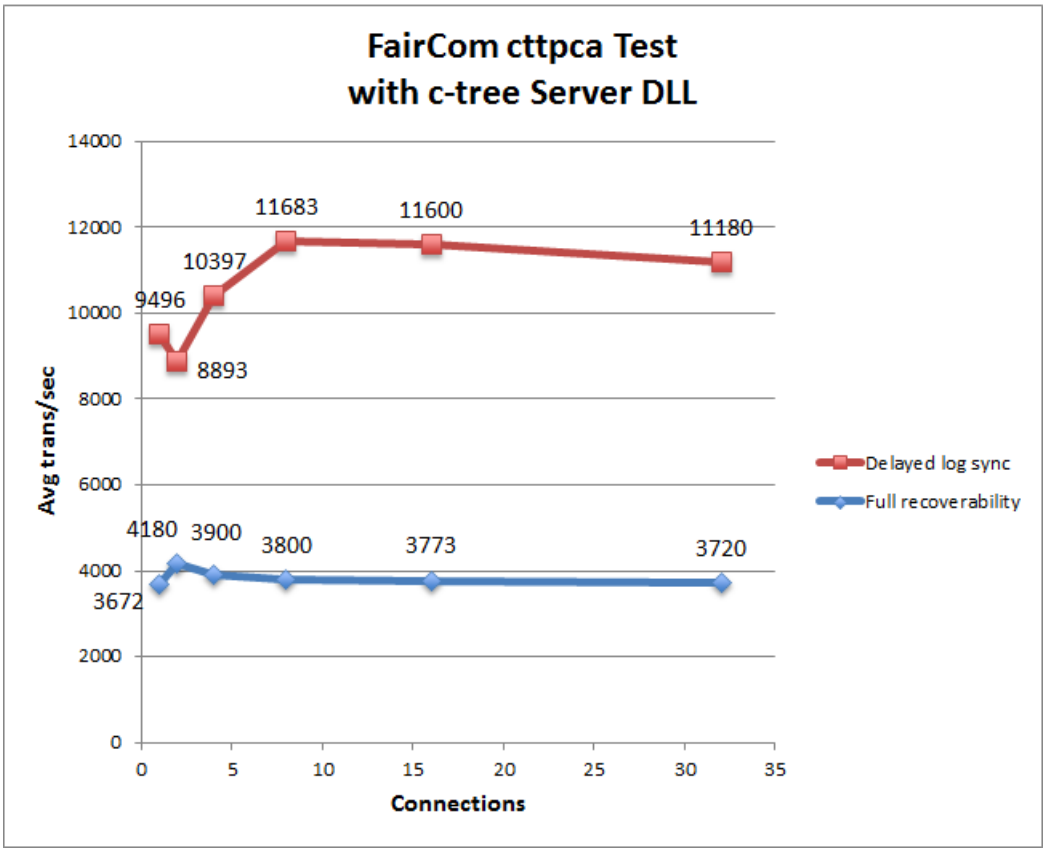
```
DELAYED_DURABILITY <N>
```

The end result can approach non-transaction performance while ensuring committed transactions to persisted storage to within less than N seconds of vulnerability. Values as low as one (1) second are shown to provide the best performance. In selected test cases, **up to 300% faster transaction throughput** has been observed. Higher values have been found to offer little, if any, additional benefit.

Performance Gains

Tests were run to compare the new Delayed Durability transaction mode with the existing transaction processing mode using `DELAYED_DURABILITY 1`, allowing the log sync to be deferred for 1 second. The results indicated that the new Delayed Durability achieved performance gains of a factor of 2 to 3 over the previous transaction processing. A setting of 1 second is recommended because it is enough for a good performance gain and higher values offer very little additional benefit.

The chart below shows performance of the new Deferred Durability transaction mode (red) and the existing standard transaction processing mode (blue):



The data was acquired using the FairCom **cttpca** test. This test was run for 30 seconds using server DLL with varying numbers of connections. New files were created before each test run. For this test, c-tree data and index caches were large enough to hold all records and index nodes.

The existing transaction processing is referred to as "full recoverability" because all transactions can be recovered. The new Deferred Durability technique can be used with *Restore Points* to allow the data to be rolled back to a known good state.

Modified Log Sync Strategy

Enabling the Delayed Durability feature activates a modified log sync strategy. Ordinarily, when a transaction commits, c-treeACE does not return from a **TRANEND()** call until log entries associated with the transaction have been written to persisted storage (e.g., your disk drive). With the new modified log sync strategy enabled only the following is ensured:

Before any data or index image is written to disk, all associated log entries are on disk.

With this modified log sync, in an extreme case, a transaction could be committed without any associated log entries on disk. Or a **TRANBEG** may be on disk, but not a corresponding **TRANEND**. This means c-tree cannot guarantee recovery of all transactions that have returned a successful commit.

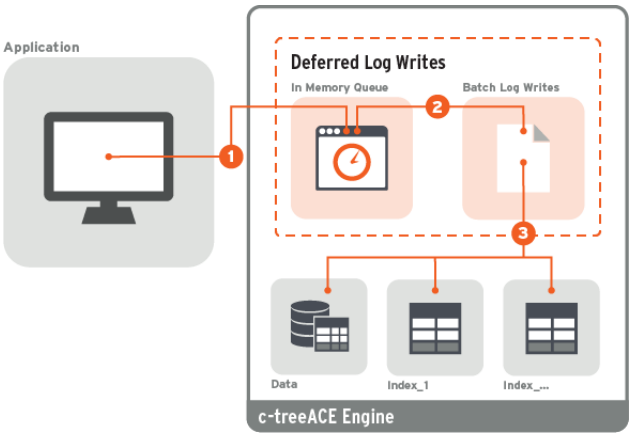
When Delayed Durability is active, the link between committing a transaction and syncing the transaction log to disk is de-coupled, and the number of log syncs is greatly reduced. Additional logic ensures that before c-tree data or index contents are written to disk, all necessary log entries are on disk. If log information is cached after a commit, the log will not necessarily be synced to persisted storage, *leaving a small vulnerable window for data loss potential in case of system failure*.

The Delayed Durability time window configuration is an added c-treeACE internal operations thread ensuring transaction log data is periodically synced on a regular basis. This log sync thread coordinates the transaction log sync to disk and guarantees that no more than N seconds can pass without a log sync to persisted storage. This thread wakes periodically and syncs accumulated transaction log data before the `DELAYED_DURABILITY` value is exceeded. A setting of 1 second is recommended as it provides excellent performance gains with minimal window of data loss in event of system failure.

Note: The transaction log buffer is always flushed when full. This buffer is defined as a 64K buffer. Any number of updates exceeding 64K will always trigger a log buffer sync to disk.

Delayed Durability Behavior

When the configuration option `DELAYED_DURABILITY` is specified in *ctsrvr.cfg* with a value greater than 0, FairCom Server does not sync its log buffer simply because a transaction commits. Instead, it syncs the log buffer if it becomes full, or if a write to the data cache requires the log buffer to be flushed to disk, or if the maximum defer time in seconds specified by the keyword is exhausted.



When this feature is in effect, a transaction commit writes to the in-memory transaction log buffer and data files in the following order:

1. **Write transaction log entries to the in-memory transaction log buffer.** The log entries are copied to the in-memory transaction log buffer and remain in the buffer until the log buffer becomes full or until a write to a transaction-controlled data or index file requires the log buffer to be written to disk. When the transaction commit call returns to the caller, there is no guarantee that the transaction log entries are on disk, and so the transaction is not guaranteed to be recoverable.
2. **Write data file updates to c-tree's data cache and/or to disk.** Depending on the options that are in effect for the file or for the database engine, this step will either write the data file updates to c-tree's data cache or write the changes to file system cache or to disk. It is not required that the changes to the data files are on disk when the transaction commits, because the transaction log entries can be used to redo the transaction if c-tree terminates abnormally before the updates to the data files are known to be on disk (which can be sometime after the transaction commits).
3. **Mark the transaction as completed.** This step causes other connections to view the key level locks as committed rather than pending commit.
4. **Unlock records.** This step allows other connections to lock and update the records that the transaction touched, if desired.

c-treeACE tracks the highest log position assigned to the contents of a data cache page for those items in preimage space that correspond to a data record. When a data-cache page is about to be written to the file system, c-tree checks if the last log byte synced to disk includes the highest required log position for the data cache page. If not, c-tree flushes the log buffer to the file system cache and syncs the file system cache to the transaction log file on disk.

Configuration Entries for Delayed Durability

`DELAYED_DURABILITY <N>` (default 0) controls whether or not to use a modified log syncing strategy.

With delayed durability enabled transaction logs are no longer sync'd to persisted storage on each commit (or other internal log buffer flush events) and instead, transaction log data is allowed to be written to the filesystem cache, and a background thread then periodically and consistently syncs transaction log contents to disk.

By allowing committed transaction entries to be written to filesystem cache and deferring the file flush can result in a very large performance gain in many cases. However, there is a trade off as a window of potential data loss vulnerability is then introduced. The period of time that transaction log contents are present in volatile filesystem cache before the flush could mean transactions already reported to the application as committed, in rare cases, might not make it to persisted storage.

With many modern storage devices there is a limited presumption that available capacitance on the system and storage device hardware that data is usually persisted even in a power outage situation, though this is not guaranteed. Thus alternate recovery strategies should be considered.

One strategy to coordinate the known state of committed database transactions with a known application state can be achieved with restore points. Restore points can be triggered by an application to create a known point in time where the application and the database are in sync. A restore point creates a special database transaction log checkpoint that can be later referenced by the application as a known good start point.

With delayed durability enabled, it is recommended to consider the use of restore points for a robust recoverable solution.

- When `DELAYED_DURABILITY` set to 0 disables delayed durability/
- When `DELAYED_DURABILITY` is set to a positive value, <*N*>, the new strategy is in use and the log sync is guaranteed to occur within <*N*> seconds. A setting of 1 second is recommended because it results in a good performance gain (higher values offer very little additional benefit). The following configuration options are set as shown below:

`SUPPRESS_LOG_FLUSH`	YES	(no idle flush of transaction files)
`SUPPRESS_LOG_SYNC`	YES	
`IDLE_TRANFLUSH`	-1	
`COMMIT_DELAY`	-1	(no commit delay)
`FORCE_LOGIDX`	ON	(all transaction indices use *ctLOGIDX*)
`COMPATIBILITY LOG_WRITETHRU`	Disabled	

Note: If the configuration file has one or more of these configuration entries set inconsistently after the `DELAYED_DURABILITY` entry, the server logs a message to *CTSTATUS.FCS* and continues to start, disabling any incompatible options after processing the configuration file.

Warning

When `DELAYED_DURABILITY` is enabled, recently committed transactions could be lost if FairCom Server terminates abnormally. For automatic recovery to succeed after FairCom Server terminates abnormally, either of the following should be considered.

1. The application must write a restore point to the log (using the **ctflush** utility or calling **ctQUIET()** with mode of *ctQTlog_restorepoint*) such that a restore point exists prior to the time the server terminated abnormally. In this case, automatic recovery recovers to that restore point.

   or

2. *ctsrvr.cfg* must contain the option `RECOVER_TO_RESTORE_POINT NO`, indicating that no restore point is needed. In this case, automatic recovery recovers to the last data that was written to the log on disk. This is the default configuration.

See Also

- *Transaction Processing Keywords /doc/ctserver/65411.htm* in the *FairCom Server Administrator's Guide* (https://docs.faircom.com/doc/*ctserver/*)

Monitoring Delayed Durability Performance

SNAPSHOT Statistics

SNAPSHOT statistics data collection related to transaction log flushing have been updated as follows:

1. The `LOG FLUSH REQUESTS` includes a “data cache” value which is the number of log flushes instigated by the Delayed Durability requirement that no data image will go to disk until the log entries associated with the image have been flushed to disk. It includes a “# checks if ...” that shows the number of times a data cache write operation checked to see if a log flush was required.
2. A new section, `DATA CACHE LOG FLUSH REQUEST DETAILS`, breaks down the data cache flush requests into the operations that trigger the flush.

Below are excerpts from three different *SNAPSHOT.FCS* files:

1. The first using `DELAYED_DURABILITY` **On**, which sets `FORCE_LOGIDX` **ON**
2. The second with `DELAYED_DURABILITY` **OFF** and `FORCE_LOGIDX` **ON**
3. The third without `DELAYED_DURABILITY` **OFF** and `FORCE_LOGIDX` **OFF**

The same single-threaded application program was run that adds 100,000 ISAM records, each with three indices, and each add is in a transaction.

Recall that when an index has a file mode with *ctLOGIDX* enabled, additional transaction log entries permit automatic recovery to repair a damaged index at the site of the damage instead of rebuilding the entire index. *ctLOGIDX* affects the log flushing dynamics.

SNAPSHOT statistics can be output with the **ctstat** Statistics Utility *-text* option.

DELAYED_DURABILITY, which sets FORCE_LOGIDX ON

```
LOG FLUSH REQUESTS
     checkpoint/endlog/commit/abort tran: 127
                               begin tran: 518
                          LOGIDX option: 20909
                        file 1st update: 20
                            replication: 0
                             data cache: 7
                                  TOTAL: 21581
```

← New statistic

```
# checks if cache write forces log flush: 1609
```

← New statistic

```
    DATA CACHE LOG FLUSH REQUEST DETAILS
                     cache aging: 0
 updated page to be re-assigned: 0
            flush on file close: 3
                      checkpoint: 0
                         CTFLUSH: 0
                     ctrbktfls(): 0
                           other: 4
```

← New section breaks down data cache value above

DELAYED_DURABILITY OFF and FORCE_LOGIDX ON

```
LOG FLUSH REQUESTS
     checkpoint/endlog/commit/abort tran: 100137
                               begin tran: 561
                          LOGIDX option: 20846
                        file 1st update: 20
                            replication: 0
                             data cache: 0
                                  TOTAL: 121564

# checks if cache write forces log flush: 0

    DATA CACHE LOG FLUSH REQUEST DETAILS
                     cache aging: 0
 updated page to be re-assigned: 0
            flush on file close: 0
                      checkpoint: 0
                         CTFLUSH: 0
                     ctrbktfls(): 0
                           other: 0
```

DELAYED_DURABILITY OFF and FORCE_LOGIDX OFF

```
LOG FLUSH REQUESTS
     checkpoint/endlog/commit/abort tran: 100135
                               begin tran: 17207
                          LOGIDX option: 0
                         file 1st update: 20
                             replication: 0
                              data cache: 0
                                   TOTAL: 117362

# checks if cache write forces log flush: 0

    DATA CACHE LOG FLUSH REQUEST DETAILS
                      cache aging: 0
 updated page to be re-assigned: 0
            flush on file close: 0
                     checkpoint: 0
                        CTFLUSH: 0
                      ctrbktfls(): 0
                            other: 0
```

Comparison

Comparing the three different *SNAPSHOT* files reveals the following:

1. The total number of log flushes is significantly lower with Delayed Durability on.
2. The reduction in total log flushes is primarily from eliminating the requirement to ensure log entries are on disk before a transaction commit can return.
3. *LOGIDX* causes a significant number of log flushes. However, when *LOGIDX* is not used, other flushing requirements imposed by indices still generate a comparable number of log flushes related to *TRANBEG* entries getting flushed.
4. The data cache impact on the number log flushes is quite small.

3.2 Time limit on flushing updated data and index cache pages for TRNLOG files

An earlier revision introduced threads that flush updated data and index cache pages for non-transaction-controlled files to ensure updates to those files were written to the file system within a configurable time limit. This revision extends this support to TRNLOG files.

The following c-tree Server configuration options set the time limit in seconds that a data cache page or index buffer can remain dirty before it is written to the file system cache. The default time limit is 60 seconds.

```
TRAN_DATA_FLUSH_SEC          <time_limit_in_seconds>
TRAN_INDEX_FLUSH_SEC         <time_limit_in_seconds>
```

- Specify `IMMEDIATE` to cause dirty pages to be written immediately.

- Specify `OFF` to disable the time limit-based flushing.

(The `NONTRAN_DATA_FLUSH_SEC` and `NONTRAN_INDEX_FLUSH_SEC` configuration options still work and apply to the non-tran flush threads.)

These options can be changed using the **ctSETCFG()** API function and using the **ctadmn** utility.

This feature causes c-tree Server to create four threads when it starts up. One thread flushes dirty pages for non-transaction data files, one thread flushes dirty buffers for non-transaction index files, one thread flushes dirty pages for TRNLOG data files, and one thread flushes dirty buffers for TRNLOG index files. The threads always exist, even if the feature is disabled.

If the time limit is set to `OFF`, the threads simply wait for the time limit to change.

When the time limit is enabled, each thread reads its updated page list from oldest to newest entry: we added a tail pointer to the update lists for this purpose; and we added a field to hold the timestamp of the first update to each page. Based on the timestamp of the oldest page and the number of entries in the oldest two counter buckets (if they are being used), we decide how long to defer after each 10 pages that we flush.

When this feature is enabled at compile time, the idle flush thread feature is disabled.

When this feature is enabled at runtime, aging for updated TRNLOG file data cache pages and index buffers is disabled.

Snapshot Changes

In V11 and later, fields have been added to the system snapshot structure (*ctGSMS*) to hold the background TRNLOG flush settings and statistics. These values are now included in the text snapshot. The ctGSMS structure version has been changed from 17 to 18 to indicate the presence of these fields. These new fields, and the fields that were added for non-tran flush feature, are defined as fields in the *bgflss[]* array of structures in *ctGSMS*.

Non-TRNLOG background file flush fields:

```
type  name                        description
----  ----                        -----------
LONG  bgflss[BGFLSNT].sctntdlmt   time limit for dirty non-transaction data pages
LONG  bgflss[BGFLSNT].sctntdmax   highwater time for dirty non-transaction data page
LONG  bgflss[BGFLSNT].sctntdage   current oldest dirty non-transaction data page
LONG8 bgflss[BGFLSNT].sctntdfls   number of non-transaction data page flushes
LONG  bgflss[BGFLSNT].sctntdbkt   non-transaction data page counter buckets (nominal)
LONG  bgflss[BGFLSNT].sctntdbkc   non-transaction data page counter buckets (current)

LONG  bgflss[BGFLSNT].sctntilmt   time limit for dirty non-transaction index buffers
LONG  bgflss[BGFLSNT].sctntimax   highwater time for dirty non-transaction index buffer
LONG  bgflss[BGFLSNT].sctntiage   current oldest dirty non-transaction index buffer
LONG8 bgflss[BGFLSNT].sctntifls   number of non-transaction index buffer flushes
LONG  bgflss[BGFLSNT].sctntibkt   non-transaction index buffer counter buckets (nominal)
LONG  bgflss[BGFLSNT].sctntibkc   non-transaction index buffer counter buckets (current)
```

TRNLOG background file flush fields:

```
type  name                        description
----  ----                        -----------
LONG  bgflss[BGFLSTR].sctntdlmt   time limit for dirty TRNLOG data pages
LONG  bgflss[BGFLSTR].sctntdmax   highwater time for dirty TRNLOG data page
LONG  bgflss[BGFLSTR].sctntdage   current oldest dirty TRNLOG data page
LONG8 bgflss[BGFLSTR].sctntdfls   number of TRNLOG data page flushes
LONG  bgflss[BGFLSTR].sctntdbkt   TRNLOG data page counter buckets (nominal)
LONG  bgflss[BGFLSTR].sctntdbkc   TRNLOG data page counter buckets (current)

LONG  bgflss[BGFLSTR].sctntilmt   time limit for dirty TRNLOG index buffers
LONG  bgflss[BGFLSTR].sctntimax   highwater time for dirty TRNLOG index buffer
LONG  bgflss[BGFLSTR].sctntiage   current oldest dirty TRNLOG index buffer
LONG8 bgflss[BGFLSTR].sctntifls   number of TRNLOG index buffer flushes
LONG  bgflss[BGFLSTR].sctntibkt   TRNLOG index buffer counter buckets (nominal)
LONG  bgflss[BGFLSTR].sctntibkc   TRNLOG index buffer counter buckets (current)
```

A text snapshot includes these values in the *SNAPSHOT.FCS* file, and the **ctsnpr** utility has been updated so that it can parse this new *SNAPSHOT.FCS* file format.

Counter Buckets

These c-tree Server configuration options set the number of counter buckets for the dirty data page and index buffer lists:

```
TRAN_DATA_FLUSH_BUCKETS      <number_of_buckets>
TRAN_INDEX_FLUSH_BUCKETS     <number_of_buckets>
```

The default number of counter buckets is 10. Setting the option to zero disables the use of the counter buckets.

(The `NONTRAN_DATA_FLUSH_BUCKETS` and `NONTRAN_INDEX_FLUSH_BUCKETS` configuration options still work and apply to the non-tran flush threads.)

Other Changes

The configuration option `DIAGNOSTICS NONTRAN_FLUSH` has been changed to `DIAGNOSTICS BACKGROUND_FLUSH`. This option enables logging of flush thread operations to the file *BGFLS.FCS*.

The configuration option `DIAGNOSTICS NONTRAN_FLUSH_BUCKETS` has been changed to `DIAGNOSTICS BACKGROUND_FLUSH_BUCKETS`. This option enables logging of flush counter bucket statistics to the file *BGFLSBKT.FCS*. Each time a text snapshot is written to the file *SNAPSHOT.FCS*, the bucket statistics are written to the file *BGFLSBKT.FCS*.

3.3 Controls for Performance AND Safety of Non-Transaction Updates

(In this discussion, a cache page that has been updated and has not yet been written to the file system is called a "dirty page.")

c-treeACE offers multiple levels of transaction protection for your data. Some applications do not require the recoverability full transaction provides for performance reasons. However, these applications may be vulnerable to data loss should system failure occur. If FairCom Server terminates abnormally, updates to data and index files that are not under full transaction control are lost if those updates have not yet been written from c-tree's in-memory data and index caches to the file system. The following factors typically reduce the number of dirty pages that exist:

1. When an updated cache page is being reused, the updated page is written to the file system cache.
2. When all connections close a c-tree file, FairCom Server writes the updated pages to the file system cache before closing the file.
3. An internal thread periodically checks if FairCom Server is idle, and if so it writes updated pages to the file system cache.

However, the combination of using very large data and index caches, keeping files open for long periods of time, and having constant activity on the system increases likelihood that more dirty cache pages exist.

It is possible to define a vulnerability window limiting potential loss of updates for your non-transaction data and index files. FairCom Server supports options to write dirty cache pages to the file system within a specified time period. This means that no more than a set amount of time can pass where data is not flushed to disk.

The following FairCom Server configuration options set the time limit (in seconds) that a data cache page or index buffer can remain dirty before it is written to the file system cache. The default time limit is `IMMEDIATE` (flush updates for non-tran files to the file system as soon as possible). Specify a value to set a time limit in seconds. Specify `OFF` to disable time limit-based flushing.

```
NONTRAN_DATA_FLUSH_SEC    <time_limit_in_seconds>
NONTRAN_INDEX_FLUSH_SEC   <time_limit_in_seconds>
```

These options can also be changed using the **ctSETCFG()** API function and using the **ctadmn** utility.

Monitoring Non-Transaction Data Flush

Fields have been added to the system snapshot structure (*ctGSMS*) to hold the non-tran flush settings and statistics. See *Time limit on flushing updated data and index cache pages for TRNLOG files* (page 33) *in the c-treeACE Programmer's Reference* (https://docs.faircom.com/doc/*ctreeplus/*).

Tuning Non-Transaction Data Flush

These FairCom Server configuration options set the number of counter buckets for the dirty data page and index buffer lists:

```
NONTRAN_DATA_FLUSH_BUCKETS   <number_of_buckets>
NONTRAN_INDEX_FLUSH_BUCKETS  <number_of_buckets>
```

The default number of counter buckets is 10. Setting the option to zero disables the use of the counter buckets.

Non-Transaction Flush Diagnostics

The configuration option `DIAGNOSTICS BACKGROUND_FLUSH` can be used to enable logging of flush thread operations to the file *NTFLS.FCS*.

The configuration option `DIAGNOSTICS BACKGROUND_FLUSH_BUCKETS` can be used to enable logging of flush counter bucket statistics to the file *NTFLSBKT.FCS*. Each time a text snapshot is written to the file *SNAPSHOT.FCS* file, the bucket statistics are written to the *NTFLSBKT.FCS* file.

3.4 Linux Direct I/O (UNBUFFERED I/O) Performance Support

In V11 and later, support for direct I/O has been enabled on Linux systems. A value of 512-bytes is used for size and alignment for direct I/O.

This feature supports both c-tree data and index files, as well as transaction logs. Configuration options are provided for both.

- `UNBUFFERED_IO` *filename* (enables direct I/O for the specified file; the filename can include wildcards, such as **.dat*)
- `UNBUFFERED_LOG_IO YES` (enables direct I/O for the transaction logs).

Note: This feature requires Linux kernel version 2.6 or later, FairCom Server logs an error message to *CTSTATUS.FCS* if these options are used on pre-2.6 Linux kernel systems. The error messages are:

```
The UNBUFFERED_IO option requires Linux kernel version 2.6 or later

The UNBUFFERED_LOG_IO option requires Linux kernel version 2.6 or later
```

3.5 COMMIT_DELAY Configuration Now Defaults to 1 ms on Linux Systems

The default FairCom Server configuration file, *ctsrvr.cfg*, included in c-treeACE Linux packages has been modified to improve performance. The `COMMIT_DELAY` setting now defaults to a value of 1 (ms). (*Previously, COMMIT_DELAY was disabled for Linux platforms, see note below.*) One ms has been shown to produce much better performance with *TRNLOG* files on Linux systems.

Note: `COMMIT_DELAY` had been disabled in an earlier correction for a Linux bug, which was preventing synchronous writes from being flushed to disk. That setting degraded *TRNLOG* performance under those circumstances and a default setting of 1 ms is again recommended with c-treeACE V10.3.1.21834 (Build 140307) or later.

3.6 Faster Open of FairCom DB API Tables

While profiling client/server performance over a network several areas in FairCom DB API file open logic were discovered to make an excessive number of c-tree API calls requiring additional round trips to the server.

In particular, **SYSCFG()**, **GETDODA()**, and **GETNAM()** calls were removed from **ctdbOpenTable(),** and internally while obtaining the path separator, greatly improving file open performance, especially when opening many tables. Logic in **ctdbUnlockTable()** was also optimized improving additional performance.

3.7 Data Record Compression Optimization

Data compression, introduced in V10, has enjoyed widespread usage with today's increased data volumes. Continued focus on this popular option highlighted several areas of improvement, including performance. One such area was faster reading of uncompressed records from a file employing data record compression.

FairCom also identified areas for improved sRLE decompression performance. Two different techniques were investigated, and each providing improvement. We expect to see a positive performance impact from these changes.

3.8 Use of Domain Sockets for Faster Unix/Linux Shared Memory Connections

In release V11 and later, Unix and Linux c-treeACE Servers use a Unix domain socket instead of named pipes for the initial shared memory protocol communication between the client and server. (*Prior to this revision, all Unix and Linux systems except AIX used named pipes for the initial shared memory connection.*)

The following shared memory protocol changes were enacted:

- Now FairCom Server uses a Unix domain socket instead of a pair of named pipes for the initial communication when a client connects to the server. FairCom Server still creates the named pipes, and when a client connects, the server waits for the client to write to either the socket or the named pipe. In this way, the server is able to support both clients that use the new method and those that use the original method.
- The `COMPATIBILITY SHMEM_PIPE` configuration option has been deprecated and no longer has any effect.

- c-treeACE clients (both ISAM and SQL) now use the Unix domain socket method when connecting using the shared memory protocol if the server indicates that it supports it. If not, the clients use the original method.
- A c-treeACE client library can be compiled with `#define NO_ctFeatUNIX_SHMEMsocket` to force the client to use the original method only.

3.9 Improved Unix Performance with Shorter Adaptive Defer for Index Node Retrievals

Index node retrieval has been modified to improve c-treeACE on Unix performance by disabling the shorter adaptive sleep (Solaris and AIX). This shorter adaptive sleep is still active for Windows systems, where it continues to yield performance gains.

3.10 Improved Performance of Failed Batch Inserts with Savepoint Restores

This enhancement improves performance of batch insert operations on a transaction-controlled file when a number of inserts succeed but then one fails (say with a duplicate key error), especially when the data file has many indexes. This situation is equivalent to starting a transaction, establishing a savepoint, adding a number of records, then adding a record that fails with a duplicate key error, then restoring to the savepoint. The savepoint restore took a long time because of disk activity required to guarantee proper recoverability. A new strategy has been implemented that avoids most of the disk activity while guaranteeing recoverability.

3.11 Improved Index Update Performance

Cleanup of key level marks in leaf nodes is now skipped on index updates resulting in substantial performance improvements during this operation.

3.12 SQL Speed-Up

SQL Optimizer Improvements

Continuous efforts have been made to improve optimizer strategies for execution plan construction. An overall enhanced engine brings numerous new features (page 56) and performance to your existing No+SQL applications.

Tables Open Faster

Table Open/Close logic has been optimized eliminating a number of unnecessary work by the server This results in far less network traffic when opening tables, which yields a dramatic performance improvement over higher latency networks.

Tables Unlock Faster

Table unlock logic has been optimized improving overall SQL performance.

Indexes Update Quicker

A new compile-time option causes certain cleanup operations to be skipped on an index update resulting in a substantial performance improvement during this operation.

Large Number of Values in an IN List

The logic used when processing a query containing a very large number of constant entries in an IN operator has been thoroughly analyzed and changes have been made to improve processing. New logic speeds up execution when the number of entries for the IN operator is large (>1000).

Reduced Table Scans

New logic avoids unnecessary table scans when scan condition are known to be false based on possible field values.

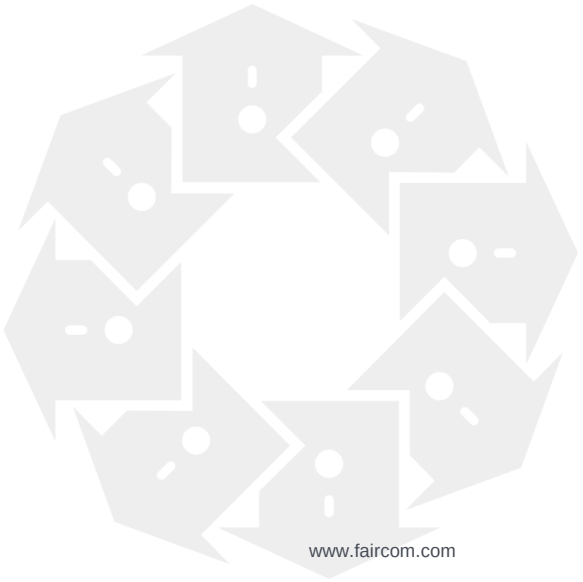
Table Locks

Table Locks (page 64) prevent other connections from reading and/or writing to the file. These can greatly reduce resource usage and contention with other connected users.

4. c-treeACE Replication Solutions

Replication has grown into a full-fledged enterprise solution with c-treeACE V11.

This section highlights many new features and additions based our experience with this immensely popular FairCom® Advanced Module Series add-on.



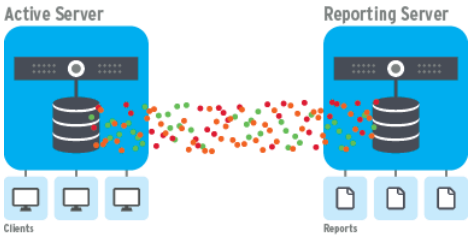
4.1 Replication for Every Need

Replication Solutions

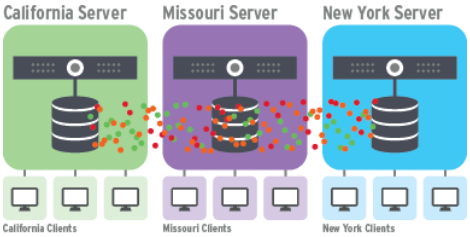
Replication provides a valuable means for maintaining data availability and business continuity. The value of c-treeACE replication can be measured in the cost of recreating data lost in case of disaster. The Replication Agent maintains an up-to-date copy of your data at a second location. Due to the low-latency algorithms used, your backup copy of the data does not need to be on-premises: the Replication Agent can maintain a duplicate of your data at a remote location connected only be a wide-area network.

Consider these various data challenges:

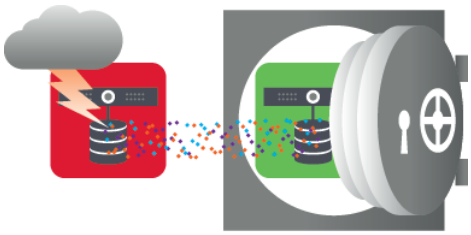
Reporting and data warehousing



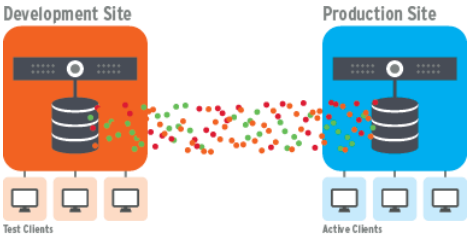
Distributed data for horizontal scaling



Failover solutions for disaster recovery and high availability



Replica environments for testing and validation



c-treeACE replication is an ideal solution for all of these purposes.

Operational Models

Within each solution, choose from a variety of topologies and replication models. c-treeACE replication is flexible enough to handle many types of environment configurations:

- **Star (Hub & Spoke)** – Recommended for multiple systems replicating to and from a central server. Local replicas can be in the same building or across the world.
- **Bidirectional** - Multiple c-treeACE Servers can be synchronized with very little latency, allowing c-treeACE to act as a distributed database.
- **Cloud Based** - When cloud is used as the host for the main source server, sites distributed around the world can replicate to a single source

Choose c-treeACE replication when you need distributed database access, high availability, and fault tolerant solutions.

4.2 Replication Manager

Introducing FairCom Replication Manager for advanced c-treeACE replication management.

FairCom Replication Manager gives you advanced, integrated control over your replication solution. Even the most complex distributed data topologies are handled with ease.

This advanced tool locates available FairCom servers on your local network and presents them in an easy-to-manage graphical display. Remote sites can also be viewed: Simply connect to your remote site and the tool will do the rest.

New to replication? No problem. Setting up and managing a replication solution is as easy as dragging and dropping. The FairCom Replication Manager provides templates you can apply to configure your entire system with just a few clicks.

Control Your Entire Replication Environment with a Single Tool

The FairCom Replication Manager presents your entire system in an intuitive graphical display. This powerful tool is browser-based so it can be run from anywhere on the network.

FairCom servers on your local and remote networks are displayed in a comprehensive network diagram. All you need to know is how to connect to them. Once connected, the tool does the rest.

Right-click on any node depicted in the tool and select the type of replication desired: Source, Target, or Bidirectional.

If you need to run a backup on a node, simply right-click it to pause, backup, and resume replication.

Disaster recovery is virtually automatic with c-treeACE. When manual intervention is required, the tool allows you to do it from your central point of administration.

Need to roll back or roll forward? Just right-click the node and select the option.

Finally, the entire replication environment is under your control from a single administrator console.

Configure the Smart Way

Select from templates or customize to suit your needs. FairCom has been analyzing and fine-tuning its replication offering to develop the perfect solution. We have packed that knowledge into the Replication Manager so it is available to everyone, from first-time users to advanced system administrators.

Templates

The secret is a set of templates that allow you to configure your system at the click of a button. Simply select the replication model that suits your needs and apply the template to the nodes in your system. Smart Templates are provided for all the most useful models (single directional, bidirectional, dual-node, star topology, etc.).

FairCom's templates take care of the rest. They don’t just fit your configuration into a preset mold; they fine-tune your system to get the optimum performance based on the characteristics of the network, servers, and other parameters.

What could be smarter than that?

Expert Mode

Particular about your system configuration? The FairCom Replication Manager is just for you. When you are ready to move beyond the templates provided by the tool, it allows you to fine-tune the entire system from a single point of administration.

Focus your attention on fine-tuning the system. Let the tool worry about making the configuration files.

Replication Agent - File Notification Queue Events

The notification process is designed to create several event actions for the same file. Before creating an action, we are checking if any actions are pending for the file. The rules are as follows:

ADD:

- if there is a pending DELETE, update it into CHANGE
- if there is a pending ADD, update the parameters
- if there is a pending CHANGE, create the new event (it is not supposed to happen)

DELETE:

- if there is a pending DELETE, ignore it
- if there is a pending ADD, remove it
- if there is a pending CHANGE, remove it and create a the new event

CHANGE:

- if there is a pending DELETE, create the new event
- if there is a pending ADD, update the parameters
- if there is a pending CHANGE, update the parameters

Replication Agent - c-tree DB Engine notification

In V11 and later, this Replication Agent configuration keyword enables monitoring of changes to the c-treeACE "engine":

```
DBENGINE_CHECK = 1
```

Enables tracking changes to the *ctsrvr.cfg*, *CTSTATUS.FCS*, **.lib*, and **.dll*.

Schedules a future action for double checking the FairCom DB API Engine changes. It is scheduled with a delay of 5 seconds by default, and all the pending checks for a given c-tree installation are grouped to avoid several re-checks. The action execution is able to change an existing installation, add a new one, or delete an old one.

4.3 Replication Agent - The Next Generation

Simplified Configuration

With V11, great effort has gone into simplifying the process of installing and configuring the FairCom® Replication Agent. A graphical editor is now available to configure the Replication Agent. It allows you to define your configuration by selecting keywords from a list. Changes to configuration files are propagated to all affected nodes on your network.

For the ultimate in control, the FairCom Replication Agent is compatible with the FairCom Replication Manager. This tool presents a graphical representation of the c-treeACE servers on your network...both local and remote. You can apply a Smart Template to configure and fine-tune any of the standard replication models. Or you can use it to dig down and edit the configuration to your exact specifications. All with just a few clicks of the mouse.

Faster

Performance improvements are constantly ongoing at FairCom. This release of the FairCom Replication Agent makes great strides in achieving the ideal of simultaneous updates to the source and target databases.

Multi-threaded DLL

The Replication Agent makes a big step forward with a new multi-threaded DLL. This enhanced architecture improves performance in demanding applications, such as heavy traffic and multiple replicated nodes.

Event Notification

The Replication Agent now supports a configuration option that causes it to run an external program when any of a list of events occurs. The program can be an executable, Windows batch file, or Unix shell script. For example, a program can be run when the Replication Agent starts up, shuts down, loses connection, etc.

On Windows the Replication Agent installs an exception handler so it can notify the external process if an unhandled exception occurs.

Directly Set a Replication Agent Start Position

The Replication Agent supports setting its starting position on the source server in the Replication Agent configuration file, *ctreplagent.cfg*. Use the `start_position` configuration option. The option accepts three possible formats:

```
start_position #current
start_position <log number> <log position> <minimum log>
start_position <log number> <log position> <last commit log> <last commit position>
```

Examples

Use the source server's current transaction log position and minimum log:

```
start_position  #current
```

```
```

Start with transaction log 10 at offset 62697158, and set minimum required log to 7:

```
start_position  10 62697158 7
```

The Replication API function, **ctReplAgentSetConfigOption()**, can be used to set the position. For example:

```
ctReplAgentSetConfigOption(ctRAOPT_start_position, "10 62697158 7", &repstt, lineno, configfile);
```

Associate Replication to Specific Server Nodes

A replicated server node identification can be specified to register a specific Replication Agent context with an associated server. The `REPL_NODEID` server configuration is used to set this identification value.

Replication Agent now supports configuration options to force required node IDs for source and target servers. These options are set with the following options in *ctreplagent.cfg:*

```
source_nodeid <dotted_notation>
target_nodeid <dotted_notation>
```

Note: Node IDs are expected to be in IPv4 address format, although actual values are arbitrary and do not need to match any actual IP server address. The main requirement is that node IDs be unique among all servers participating in replication.

Exception Handling

When this option is used for a source and/or target server, the Replication Agent requires that the server has set its node ID to that value (using the `REPL_NODEID` configuration option in *ctsrvr.cfg*). If the server has not set the value, the Replication Agent logs the following message to *ctreplagent.log*:

```
Node ID is required for source server but it is not set
```

If an error occurs retrieving a server node ID, the agent logs this message:

```
Node ID is required for source server but it cannot be read: error code <errorcode>
```

If a node ID mismatch is detected when Replication Agent is starting, a message is logged to *ctreplagent.log* and the agent shuts down:

```
Source server node ID is expected to be '<expectednodeid>' but is '<actualnodeid>'
```

If the node IDs match when Replication Agent starts, but the connection to the source or target server is lost and reestablished, and a node ID mismatch is detected, the Replication Agent logs an error message but does not terminate. Instead, it continues retrying the connection and checking for a proper node ID match every 5 seconds.

Replication API

The options can also be set by calling the **ctReplAgentSetConfigOption()** function.

Example

```
ctReplAgentSetConfigOption(""source_nodeid"", ""10.0.0.1"", prepstt, lineno, cfgfile);
ctReplAgentSetConfigOption(""target_nodeid"", ""10.0.0.2"", prepstt, lineno, cfgfile);
```

External Notification of Replication Agent Events

In V11 and later, the Replication Agent supports a configuration option, `notify_events yes`, that triggers execution of an external program. This external execution can be a Windows batch file, Unix shell script, or executable and is expected to be named **replmon**. When an event in the list of event codes below occurs **replmon** is called.

Two string values are passed to **replmon** upon event notification: an error code and one of the numeric event codes listed below. **replmon** can then process this exception notification as appropriate for the environment.

1. Replication Agent is starting
2. Replication Agent is shutting down
3. Replication Agent is entering paused state
4. Replication Agent is resuming operation
5. Replication Agent connected to source server
6. Replication Agent connected to target server
7. Replication Agent lost connection to source server
8. Replication Agent lost connection to target server
9. Replication Agent disconnected from source server
10. Replication Agent disconnected from target server
11. Replication Agent terminating due to exception

Example

In the example below, **replmon.bat** logs the date, time, event description, and error code to the file *replevent.log*:

```
@echo off

if "%1" == "1" (
 set evdesc=Replication agent is starting.
) else if "%1" == "2" (
 set evdesc=Replication agent is shutting down. Error code %2.
) else if "%1" == "3" (
 set evdesc=Replication agent has been paused.
) else if "%1" == "4" (
 set evdesc=Replication agent is resuming normal operation.
) else if "%1" == "5" (
 set evdesc=Replication agent connected to source server.
) else if "%1" == "6" (
 set evdesc=Replication agent connected to target server.
) else if "%1" == "7" (
 set evdesc=Replication agent lost connection to source server. Error code %2.
) else if "%1" == "8" (
 set evdesc=Replication agent lost connection to target server. Error code %2.
) else if "%1" == "9" (
 set evdesc=Replication agent disconnected from source server.
) else if "%1" == "10" (
 set evdesc=Replication agent disconnected from target server.
) else if "%1" == "11" (
 set evdesc=Replication agent terminating due to exception. Exception code %2.
) else (
 set evdesc=Replication agent event code %1. Error code %2.
)

for /f "usebackq tokens=*" %%j in (`date /t`) do set dt=%%j
for /f "usebackq tokens=*" %%j in (`time /t`) do set tm=%%j
echo %dt%%tm% %evdesc% >> replevent.log
```

On Windows the Replication Agent installs an exception handler so it can notify the external process if an unhandled exception occurs.

Configurable Timeout for Replication Agent Network Calls

A loss of network connectivity can cause the Replication Agent to hang in a socket send or receive call to a source or target server. To allow the Replication Agent to detect and recover from this situation, V11 and later now supports a configuration option to set a timeout on its send and receive calls. The following option can be specified in the Replication Agent configuration file, *ctreplagent.cfg*:

```
socket_timeout <timeout_in_seconds>
```

The default is 5 seconds. A setting of zero disables the timeout.

Correctly Terminate Orphaned Replication Agent Source and Target Server Connections

It is possible for a network, particularly a WAN, to periodically lose connectivity between the Replication Agent and its source or target servers. In this situation, the Replication Agent process may reconnect to the servers and find that the original server-side connection still exists. This causes the Replication Agent to fail to access the servers and log one of the following errors to *ctreplagent.log*.

For the source server:

```
ERR: Failed to enable replication log position persistence on data source: 780
ERR: Check for two replication agents using the same unique ID (<agentid>)
```

For the target server:

```
ERR: Failed to lock replication state record on data target. Check for two replication agents using the
same unique ID (<agentid>).
ERR: Failed to read replication state on data target: 827
```

In V11 (and added into this V10.4.1 line), this reconnection logic has been changed allowing the Replication Agent to locate the connection that is using the same unique ID and request it to terminate. When this occurs the Replication Agent now logs the following messages:

For the source server:

```
INF: Successfully disconnected replication reader using same ID on data source
```

For the target server:

```
ERR: Failed to lock replication state record on data target. Check for two replication agents using the
same unique ID (RA).
ERR: Failed to read replication state on data target: 42
INF: Successfully disconnected replication reader using same ID on data target
```

Note: This modification requires updating the Replication Agent as well source and target servers to this new line to obtain corrected support.

ctrepd Replication Debug Utility

In V11 and later, a useful replication debugging/test utility, **ctrepd.exe**, is now included in the Replication Agent package. The executable can be found in the *\tools\replication* directory.

This client utility can be built from standard c-treeACE Professional package by building either a single-user or multi-threaded client library and including samples and utilities.

ctrepd allows targeted troubleshooting, benchmarking and selected replication actions against the source server. For example, setting a log position, or log requirements of specific Replication Agent IDs.

Improved Error Handling for Replication Agent HTRN_ERR (520)

When files are copied to a target replication server, index files contain transaction high-water marks that can conflict with new transaction numbering of incoming replicated transactions. It is very possible error **HTRN_ERR** (520, high transaction mark error) may be observed in the replication exception log when this occurs. It is likely the first transaction that replicated to this file fail with this error.

The Replication Agent will now attempt to handle **HTRN_ERR** errors by aborting and retrying the transaction. Retries are not attempted if the Replication Agent is using the following option:

`exception_mode operation`.

Note: With this change, it is no longer necessary to run a **CLNIDXX** operation (using the **ctclnidxx** utility, the `!CLNIDXX` dynamic dump restore option, or the `AUTO_CLNIDXX` *ctsrvr.cfg* option) prior to accessing the target server's copy of the file.

Support Partial Record Rewrite in Local/Master Synchronous Replication

Local/master Synchronous replication is used when one server is a designated master server with multiple local servers. In this model, the master server is always updated along with the local server where the update originated. This is done in a distributed all or nothing transaction. Other local servers, not participating in the transaction, then receive the update via replication from the Replication Agent.

When replication was using the local/master model, some partial variable-length record updates were not applied to the master copy of the data. The logic has been modified to prevent this.

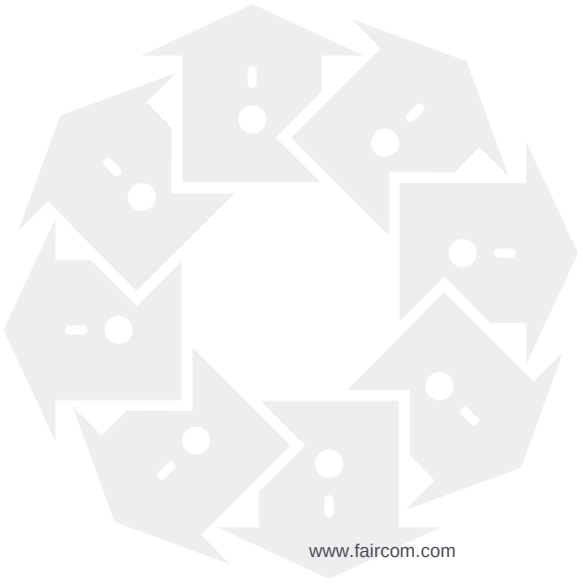
4.4 Improved Responsiveness of the c-treeACE Replication Monitor

Several changes have been made to the c-treeACE Replication Monitor to make the tool more responsive. In particular, when the replication exception log had a large number of entries and was not local, the exception log viewer was very slow. In some cases, adjusting the exception log slider bar also seemed to trigger a refresh of the record buffers. Several enhancements have been made to make the tool more responsive.

The changes include a new **Paging** control in the exception log tab to reduce the number of records in the data grid to 1000 or 10000.

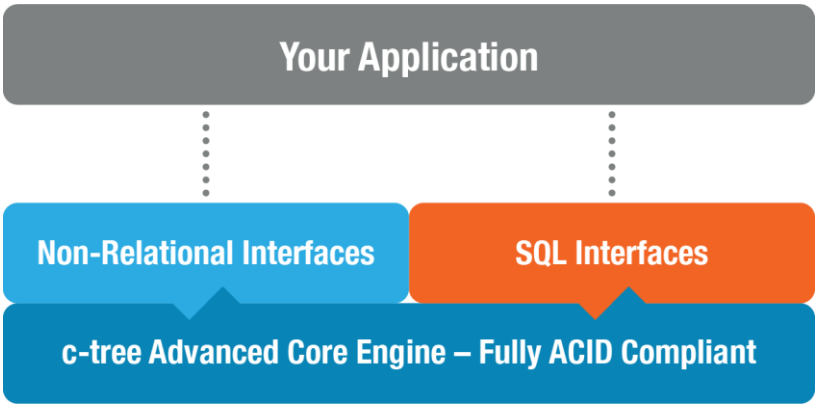
5. No+SQL Data Access

FairCom's dedicated focus on database engineering, teamed with unique customer data challenges, results in new and novel SQL access to existing data formats and bring SQL access to your existing data challenges.



5.1 c-treeACE No+SQL Solutions

FairCom refused to accept the "either/or" proposition of SQL versus NoSQL data structures. Instead, we have engineered a "No+SQL" architecture, providing unique solutions to challenging database problems.



Let FairCom help discover solutions to your No+SQL challenge.

5.2 c-treeACE SQL FILESET for Dynamic Joining of Physical Data Files

FairCom takes the lead in bridging the gap between SQL and the more-efficient direct record access.

Much of today's database industry news is filled with terms such as "Big Data" and NoSL and rest assured, FairCom maintains an active watch on these new data frontiers.

We regularly hear of datasets measured in hundreds of terabytes and even petabytes. Realizing the value of data, you've possibly accumulated thousands of files over the years and now you require an efficient query capability across those file.

c-treeACE SQL has the ability to "import" all your tables into a complete modern SQL interface. However, in many situations, traditional SQL views are simply not feasible due to the sheer number of tables involved.

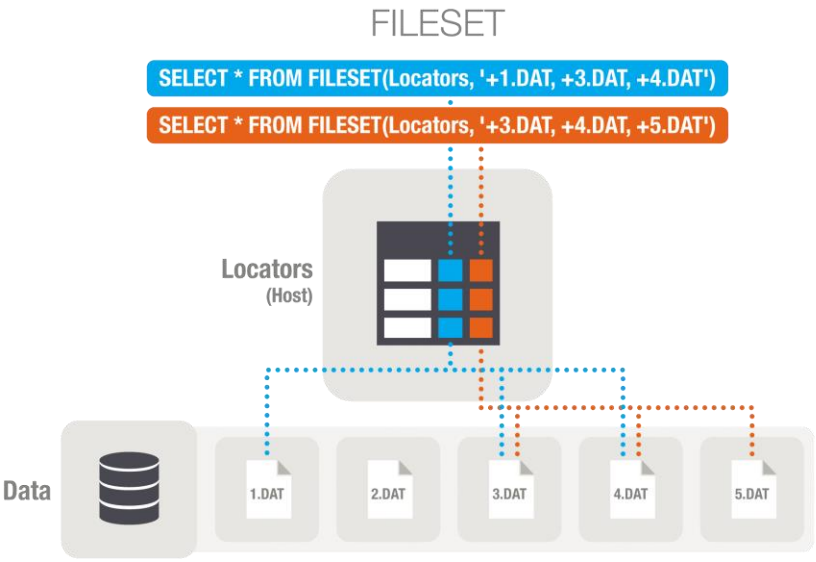
Consider what your "big data" challenge may be:

- Data files are frequently created “on-the-fly” by the application, sometimes daily.
- Traditional SQL queries require static SQL dictionary management consisting of defined entities (i.e., all entities are required to be present in a single SQL Dictionary).
- Performance concerns arise when building SQL views over hundreds to thousands of tables.

"How do I sort through all of this accumulated data from my very successful c-treeACE application"?

FairCom enjoys the challenge and introduces FILESET, a broadened query capability bringing modern advanced query to your existing volumes of data.

Our new FILESET feature allows SQL queries directly over multiple external c-tree files. This new c-treeACE SQL extension brings advanced data access for BI, statistical analysis and other complex query and reporting needs.



Introducing Dynamic FILESET

FILESET allows you to operate on a number of files as a single source when making SQL queries. Realizing data was already "partitioned", FairCom engineers looked in the opposite direction and created partitioned tables on-the-fly!

A single master host table is defined using an established schema of existing tables. A new FILESET host creation utility (page 227) is available allowing developers to point to an existing data table and create a fully functional host table with all required attributes.

At query time, this host table is referenced with a sequence of physical table names to produce a single FILESET™ table. Using dynamic partitioning techniques from our powerful partitioned file feature, c-treeACE automatically links together individual tables into a unified data view.

```
SELECT  *  FROM  FILESET(custmaster_host, '+customer2013.dat+custmaster2014.dat+custmaster2015.dat');
```

With FairCom's new SQL FILESET™ feature, SQL statements can be defined with in-line tables built on-the-fly from specific physical data files. Dynamically assigning files to a host table, queries can be executed directly against these tables, including sophisticated joins of multiple dynamic host tables.

c-treeACE SQL grammar has been extended allowing a dynamic list of partitions to be specified when executing queries on dynamic partitioned files. c-treeACE SQL accesses necessary files and makes them appear as a single table to SQL, thereby eliminating the overhead of creating many SQL views over a large number of files. A FILESET is created dynamically—on-the-fly—such that SQL sees the results as a simple, static table.

Find your “Needle in a Haystack”

The FILESET concept can be used to simplify SQL queries. Rather than defining complex views or writing a complex query across multiple files—possibly thousands—FILESETs allow you to create a simple query as though you were searching only a single table. A new function allows you to define a list of dynamic partition members such that you can set the partition table dynamic members when using FILESETs.

Syntax

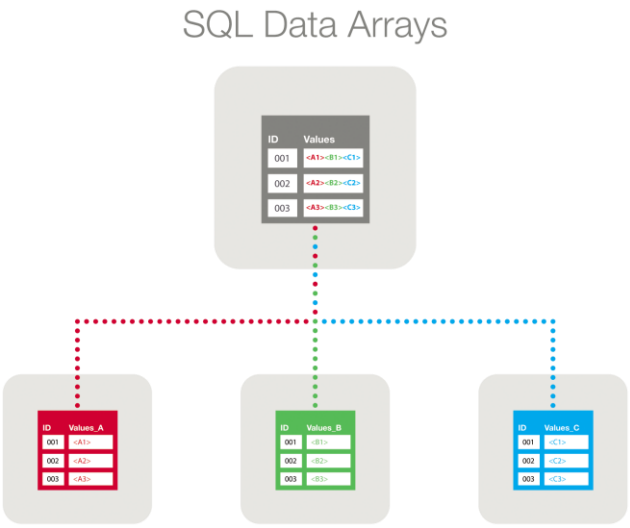
```
FILESET  ( hosttablename, string_of_physical_files )
    string_of_physical_files
            [ separator_char
```

Example

```
SELECT  name, zipcode, data
    FROM  FILESET(hostmaster, '+/home/data/volume1.dat+/home/data/volume2.dat+/home/data/volume3.dat')
    WHERE data > 99
    ORDER BY zipcode
```

5.3 c-treeACE SQL Data Arrays™ for Sub-Record Data

c-tree application developers have been very creative in how they take advantage of c-treeACE for performance-driven applications. One technique employed is storing many small data records in a single c-tree record. By packing multiple data points into a single write operation one can greatly boost performance. Consider the savings for rapidly accumulated time-series data. But, until now, there was no way to efficiently query that data via SQL.



By combining Multi Schema Table support (MRT, introduced in V10) with additional support to "sub-read" data points from records, c-treeACE SQL can layer new views onto this data access challenge. We call this capability SQL Data Arrays.

Given appropriate schemas for each sub-record type, MRT definitions can be created providing virtual table views over these data points. With modifications to read routines in FairCom DB API, c-treeACE SQL can access individual records contained in the c-tree record structure. A simple rule associating each data point with its associated schema is all that is needed to complete this highly extensible support. Quickly view data relationships you never saw before.

Contact your nearest FairCom office for the latest availability of this feature.

5.4 Cutting-Edge No+SQL Features

Be sure to review the latest No+SQL integration features.

FILESET (page 52) for Querying Multiple Physical ISAM Files

The new FILESET syntax allows simple SQL queries to operate on thousands of files with little to no loss in performance. This new extension to SQL provides the flexibility and performance you need for challenging statistical analysis and other complex queries across multiple physical data files.

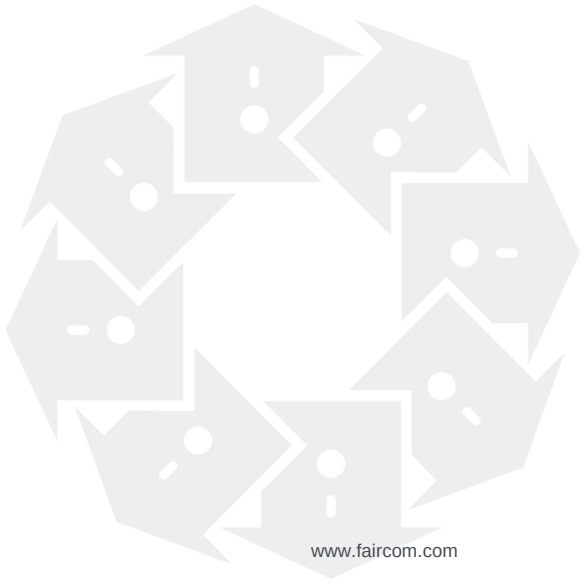
SQL Data Arrays (page 54) allow Sub-Record Reads

SQL enables support for data files with multiple record schemas where each c-tree data record contains zero or more sub-records. The main data file can be imported into a SQL database creating a multi-record table (MRT) for each sub-record type. FairCom DB API has been updated to read individual field values from a sub-record. After the MRTs are created, they can be queried through SQL.

6. c-treeACE SQL Features

V11 moves c-treeACE SQL support well beyond the capabilities of past releases, providing a robust environment for your data organization and access.

Many great new features will enhance your c-treeACE SQL applications and are described in this section.



6.1 New Stored Procedure Development Frameworks

c-treeACE SQL stored procedures (SP), user defined functions (UDF), and triggers provide powerful extensions to standard SQL operations allowing centralized business rules and enforcement of those rules.

The popularity of this feature has prompted several big new FairCom developments greatly extending development ease and platform extensibility:

- **NetBeans Plugin** - Cross-platform Java is the current framework used to develop SP, UDF, and triggers for c-treeACE SQL. To ease development, a NetBeans/Eclipse plug-in has been created to provide a complete IDE for Java procedure development. This is immediately available to users of Java stored procedures.
- **.NET Support** - Extending platform flexibility for Windows developers, c-treeACE SQL now supports stored procedures, user-defined functions, and triggers for the .NET framework. New Visual Studio integration provides simplified development of SP, UDF, and triggers in any .NET language that compiles to the CLI. .NET assembly dynamic link libraries are now easily deployed to remote servers from a convenient development platform. Templates for C# and Visual Basic are provided.
- **Dump and Deploy Utilities** - To simplify the process of deploying SP, UDF, and triggers to multiple servers, scriptable command-line utilities can now be used to dump existing SPs, UDFs, or triggers and deploy to remote servers. These are provided to assist application deployments.

6.2 SQL Group Support for User Role Management

A common problem when moving to c-treeACE SQL is managing access permissions for a number of users. Frequently, there are groups of users with a defined role requiring the same granted privileges to data. And, of course, users may need to be added or removed over time. Up until now, table privileges must be individually granted. Having a role definition defined at the group level makes it much easier to grant privileges in bulk.

Group management of users is now available for c-treeACE SQL. Assigning users to established groups greatly simplifies SQL rights management.

Group definitions are administered and maintained within c-treeACE with already existing utilities such as the **ctadmn** Administrator utility, **sa_admin** Security Administration utility or the Security Administrator tool. All user and group definitions are managed from this central point and are available from both the core c-treeACE database engine as well as from c-treeACE SQL. These definitions are securely stored in the *FAIRCOM.FCS* user group security information file.

Once a group is created, GRANT privileges to the group from SQL as you would a user. When users are assigned to the group with the mentioned utilities, these privileges are immediately applied to that user.

Important: There remains a limitation that SQL group names cannot be the same as a SQL user name.

Example

To create 4 users belonging to two group definitions you could script something such as the following:

Create Groups

```
sa_admin -aADMIN  -pADMIN  -oga ODBCUSER   -d ODBC_users
sa_admin -aADMIN  -pADMIN  -oga ORDERDESK  -d Shipping_desk
```

GRANT SQL Group Privileges

```
isql  -s grant_file.sql -u ADMIN -a ADMIN  6597@localhost:ctreeSQL
```

where grant_file.sql contains the following:

```
GRANT SELECT ON custmaster TO ODBCUSER
GRANT SELECT ON custorder  TO ODBCUSER
GRANT SELECT ON orderitem  TO ODBCUSER
GRANT SELECT ON itemmaster TO ODBCUSER

GRANT ALL ON custmaster TO ORDERDESK
GRANT ALL ON custorder  TO ORDERDESK
GRANT ALL ON orderitem  TO ORDERDESK
GRANT ALL ON itemmaster TO ORDERDESK
```

Now, any user assigned to ODBCUSER receive SELECT privilege, and likewise, users assigned to ORDERDESK. receive ALL privileges

Create Users

```
sa_admin -aADMIN -pADMIN  -oua JOEBOB    -w secret_password -g ODBCUSER
sa_admin -aADMIN -pADMIN  -oua SALLYSUE  -w secret_password -g ODBCUSER

sa_admin -aADMIN -pADMIN  -oua ALICE     -w secret_password -g ORDERDESK
sa_admin -aADMIN -pADMIN  -oua WALTER    -w secret_password -g ORDERDESK
```

Further privileges can be directly assigned to users on top of any assigned group privileges:

```
GRANT EXECUTE ON update_monthly_balance_proc TO ALICE
```

6.3 c-treeACE SQL Entity Framework 6 Support with ADO.NET

Microsoft added numerous enhancements with Entity Framework 6, EF6. It is now a standalone package that can be added to other projects as needed. Enhancements include faster opening of context objects when many tables and relationships are defined. The framework can be customized, and code and requests can be configured.

The c-treeACE SQL ADO.NET Provider has been updated to support the latest Entity Framework, 6. EF6 is compatible with Microsoft .NET Framework 4 and Visual Studio 2010 and 2013.

6.4 Table Valued Functions

Table valued functions make dynamic table query available by allowing calculated stored procedure results to be used in place of a table reference in the FROM clause of a SELECT statement.

Example

Consider a stored procedure that builds a list of customers given a set of parameters:

```
proc_get_overdue_customers_by_zip (
    IN zip CHAR(5),
    IN  calc_by_days INTEGER
)
RESULT (last_name CHAR(20), first_name CHAR(20), balance_due MONEY, days_overdue INTEGER)
```

As a table valued function, this procedure can be directly referenced in your SELECT statement.

```
SELECT  last_name, first_name, balance_due, days_overdue  FROM
proc_get_overdue_customers_by_zip('65203', 90);
```

6.5 Extended ALTER VIEW, ALTER TABLE, and ALTER INDEX Flexibility

ALTER functionality has been extended for multiple SQL actions, including views, indexes and tables.

ALTER VIEW

Now, an existing view can be replaced with a new view definition, retaining existing view permissions (which would be lost with DROP VIEW/CREATE VIEW).

Example

```
ALTER VIEW customers_by_zip AS SELECT cmname, cmaddress FROM custmaster  ORDER BY cmzipcode;
```

ALTER INDEX for Renaming Indexes

ALTER INDEX now supports renaming an index or adding a new storage attribute such as partitioning

Example

```
ALTER INDEX oi_ordrnumb_idx ON ordernum RENAME TO new_oi_ordrnumb_idx;
ALTER INDEX co_ordrdate_partidx ON custordr STORAGE_ATTRIBUTES 'partition=QUARTER(co_ordrdate)';
```

ALTER TABLE for Renaming Tables

ALTER TABLE now supports renaming tables, columns, and constraints.

Examples

```
ALTER TABLE  custmaster RENAME TO archive_custmaster;
ALTER TABLE  custmaster RENAME COLUMN cmzipcode TO cmzipcode_plusseven;
ALTER TABLE  orderitem  RENAME CONSTRAINT oi_ordrnumb TO new_oi_ordrnumb;
```

6.6 Common Table Expressions (CTE) and Recursive Queries

Recursive Queries are useful for querying hierarchical data such as tree structured data using a common table expression (CTE). A CTE can be called repeatedly (recursively), fetching a subset of the results each time until it has built an entire result set. A good example is the way a hierarchy, such as a family tree, can be extracted from a database by recursively calling a CTE that gets all children of a specified parent. They are also a useful technique to "flatten" data from multi-row queries.

Example 1: Count Items

This example shows a way to count items:

```
WITH RECURSIVE counter (n) AS (
  SELECT 1 as n
  UNION ALL
  SELECT n + 1 FROM counter WHERE n < 10
)
SELECT * from counter;
```

The output from this example would be:

```
  n|

  1|
  2|
  3|
  4|
  5|
  6|
  7|
  8|
  9|
 10|

10 record(s) returned
```

Example 2:

```
CREATE TABLE testtable (id int, name char(200));
insert into testtable values (1,'First');
insert into testtable values (2, 'Second');
insert into testtable values (3, 'Third');
WITH RECURSIVE fn(id, name, n)
AS (
    SELECT id, name, 1 as n from testtable where id = 1
    UNION ALL
    SELECT testtable.id, testtable.name, fn.n+1 from fn, testtable where testtable.id = fn.n+1
)
SELECT id, name FROM fn;
```

The output from this example would be:

```
id| name    |

 1| First   |
 2| Second  |
 3| Third   |
```

Example 3: Employee Reporting "Tree"

This example builds an employee reporting "tree":

```
WITH RECURSIVE company_tree_list ( boss_id, emp_id, title, dept_id, level ) AS
(
-- Anchor member is defined.
  SELECT emp.name, emp.emp_id, emp.title, edh.dep_id, 0 AS level
        FROM employee_list AS emp
        INNER JOIN emp_dept_history AS edh
            ON e.emp_id, = edh.entity_id AND edh.enddate IS NULL
            WHERE boss_id IS NULL
    UNION ALL
-- Recursive member is defined referencing cte_name.
  SELECT emp.boss_id, emp.emp_id, emp.title, edh.dep_id,    level + 1
        FROM employee_list AS e
        INNER JOIN emp_dept_history AS edh
```

```
            ON emp.emp_id = edh.entity_id AND edh.enddate IS NULL
        INNER JOIN company_tree_list AS d
            ON emp.boss_id = d.emp_id
)
SELECT name, emp_id, title, dept_id
    FROM company_tree_list
    INNER JOIN departments AS dept
        ON company_tree_list.dept_id = dept.dept_id
    WHERE dept.group = 'Sales' OR level = 0;
```

Example 4: Report a Category's Parent

```
create table "admin"."people" (
"id" integer,
"name" character(26),
"parent_id" integer
);

insert into "admin"."people" values('1','Root A ',NULL);
insert into "admin"."people" values('2','Root B ',NULL);
insert into "admin"."people" values('3','Child A1 ','1');
insert into "admin"."people" values('4','Child A2 ','1');
insert into "admin"."people" values('5','Child B1 ','2');
insert into "admin"."people" values('6','Child B2 ','2');
insert into "admin"."people" values('7','Grandchild A1a ','3');
insert into "admin"."people" values('8','Grandchild A1b ','3');


WITH RECURSIVE sub_tree (id, name, relative_depth ) AS (
  SELECT id, name, 1 AS relative_depth
  FROM people
  WHERE name = 'Child A1'
  UNION ALL
  SELECT cat.id, cat.name, st.relative_depth + 1
  FROM people cat,  sub_tree  st
  WHERE cat.parent_id = st.id
)
SELECT * FROM sub_tree;

id | name                      | relative_depth |
 3 | Child A1                  |              1 |
 7 | Grandchild A1a            |              2 |
 8 | Grandchild A1b            |              2 |

3 record(s) returned
```

Example 5: Flatten Multiple Rows into a Single Field

```
create table mytest (
f1 varchar (20),
f2 integer);
```

```
insert into mytest values('aaa','1');
insert into mytest values('bbb','2');
insert into mytest values('ccccc','3');
insert into mytest values('ddd','4');
insert into mytest values('eeeeeeee','5');
insert into mytest values('f','6');
insert into mytest values('gggg','7');
insert into mytest values('hhh','8');

WITH RECURSIVE myflat ( flatf1, f2) AS
(
SELECT cast(f1 as varchar(8192)), f2
FROM mytest
where f2 = 1
ORDER BY f2
UNION ALL
SELECT flatf1 || f1, mytest.f2
FROM mytest, myflat parent
where parent.f2 + 1 = mytest.f2
)
SELECT top 1 flatf1
FROM myflat order by f2 desc;

aaabbbccccddddeeeeeeeeefgggghhh |

1 record(s) returned
```

6.7 LOCK TABLE Statement Added

In applications where a large number of rows will be accessed for either reading or modifying, c-treeACE SQL provides an explicit locking construct for locking all the rows of a table. The `LOCK TABLE` statement explicitly locks a table in either `SHARE` or `EXCLUSIVE` mode.

The following example shows how to acquire a lock in the `EXCLUSIVE` mode for a table called `customer` from an ESQL command line:

```
EXEC SQL
     LOCK TABLE customer IN EXCLUSIVE MODE;
```

The above statement will prevent other transactions from either reading or modifying the table customer until the transaction either commits or performs a rollback.

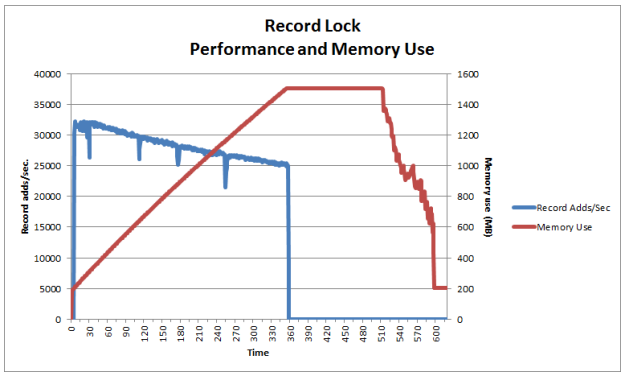
The following example shows acquiring lock in the `SHARE` mode for a table called `orders`:

```
EXEC SQL
     LOCK TABLE orders IN SHARE MODE;
```

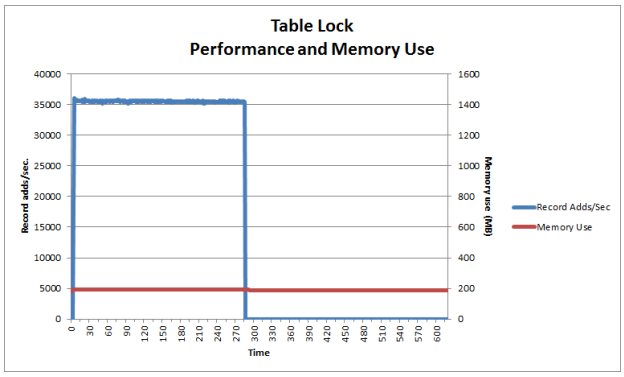
The above statement will prevent other transactions from modifying the `orders` table until the transaction either commits or performs a rollback.

Performance Gains

The charts below shows the results of a C# stored procedure adding 10 million records to a c-tree data file (non-transaction file with no index). The blue line indicates performance (records added per second) and the red line indicates memory use.



Using individual record locks, performance (blue line) decreases over time and memory use (red line) increases over time. When the adds are finished and the commit happens, noticeable time is spent freeing locks. Insert time: 356 sec.



Using table locks, record insert performance (blue line) starts higher than with individual record locks—however it stays at the same level of performance throughout. Memory use (red line) initially increases by a small amount and then stays at that level. Insert time: 284 sec (20% faster).

6.8 Scrollable SQL Cursors

JDBC supports scrollable cursors and c-treeACE SQL now implements this support.

Forward and backward cursor scrolling is supported and includes relative and absolute record number fetching.

Scrollable cursors are not enabled by default for a given connection. Scrollable cursors require that the query be executed in its entirety before the first fetch and results loaded into a temporary table. This can have preforming implications, especially for queries with a large result set where the client may only fetch the first few results.

For a JDBC connection the cursor type specified at connection time controls whether or not scrollable cursors are supported.

6.9 Unicode default charset for SQL CHAR and VARCHAR changed from US-ASCII to ISO-8859-1

Prior to V11, the c-treeACE Unicode implementation mandated US-ASCII (7-bit) chars in [VAR]CHAR fields, which may have been too strict for customers who use an 8-bit charset. For this reason, the default charset has been changed to ISO-8859-1 (aka Latin1). According to ICU:

> ISO-8859-1 is relatively unproblematic — if its limited character repertoire is sufficient — because it is converted trivially (1:1) to Unicode, avoiding conversion table problems for its small set of characters. (By contrast, proper conversion from US-ASCII requires a check for illegal byte values 0x80..0xff, which is an unnecessary complication for modern systems with 8-bit bytes. ISO-8859-1 is nearly as ubiquitous for modern systems as US-ASCII was for 7-bit systems.)

The modification should introduce no backward compatibility problems as US-ASCII is a subset of ISO-8859-1.

6.10 Allow DOUBLE as an alias for DOUBLE PRECISION data type

The c-treeACE SQL parser has been updated to support DOUBLE as an alias for DOUBLE PRECISION.

6.11 Configuration Options for c-treeACE SQL LATTE Subsystem

Configuration options are available in the server configuration file *ctsrvr.cfg* to configure the LATTE sorting subsystem for c-treeACE SQL. This subsystem is indicated as: `SUBSYSTEM SQL LATTE`. Keywords that affect this subsystem must be enclosed in curly braces as shown in the example below:

```
SUBSYSTEM SQL LATTE
{
SQL LATTE KEYWORD
}
```

This subsystem accepts the following keywords:

- `MAX_MEMORY` - Maximum advisable memory per environment. Default: 64M
- `CACHE_BLK` - Default number of blocks in the temporary cache of a table. Default: 8
- `MAX_STORE` - The maximum number of swap files (128M each) LATTE can create. This setting can be 65535 at max. Default: 128
- `NONE` - This option is used in conjunction with the tamper-proof settings file (*.set*). When specified, the entire `SUBSYSTEM SQL LATTE` cannot be overridden in the *ctsrvr.cfg* file.

Example

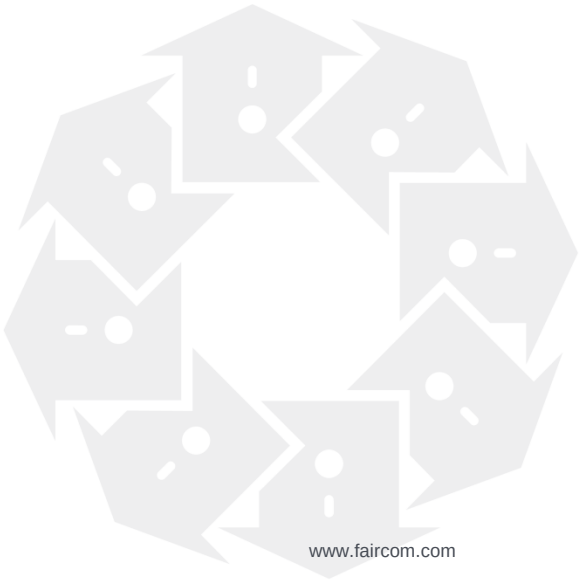
Notice that any keywords listed above must be in a section of the configuration file labeled SUBSYSTEM SQL LATTE and enclosed in curly braces. For example, include the following to set the maximum advisable memory per environment to 64MB:

```
SUBSYSTEM SQL LATTE
{
MAX_MEMORY 64M
}
```

7. Extended c-treeACE SQL Stored Procedure Frameworks

Move your core business logic server-side and enjoy big performance gains.

c-treeACE SQL stored procedure support moves beyond Java in V11 and now includes extensive .NET framework support. Java support also includes a NetBeans IDE update.



7.1 c-treeACE SQL Stored Procedure Development in the .NET Framework

Microsoft .NET framework provides a common runtime functionality for Windows applications. This allows a consistent and comprehensive programming model for Windows developers. Extending platform flexibility for Windows developers, c-treeACE SQL now supports stored procedures, user-defined functions, and triggers within the .NET framework.

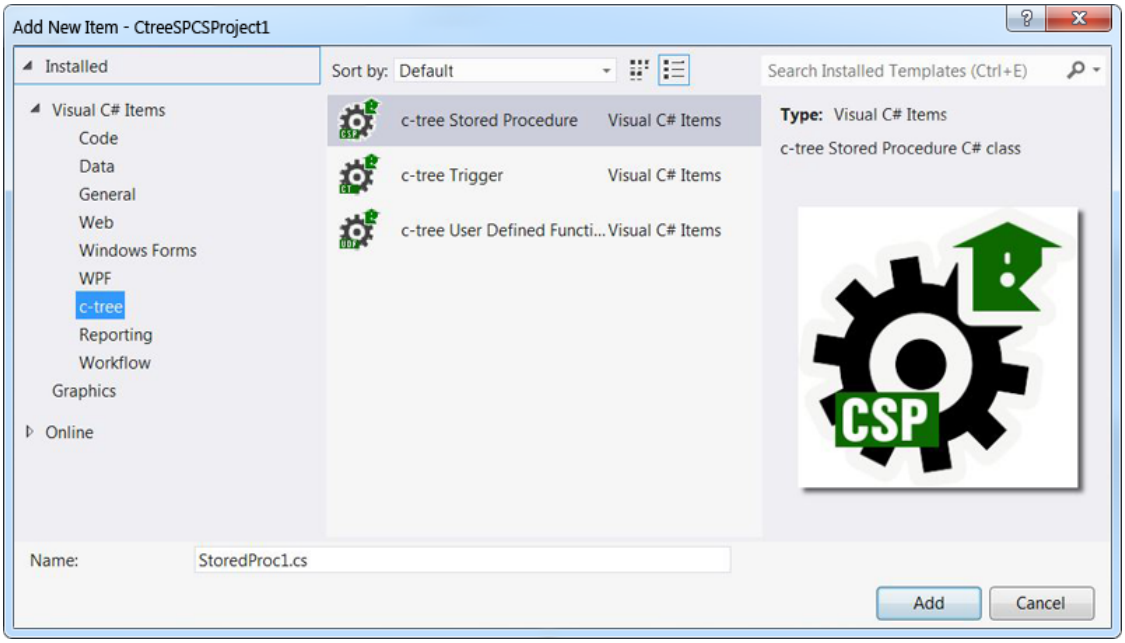
Key Features

- SP, UDF, and triggers can be written in any language that compiles to a CLI
- Exposes SQL classes similar to ADO
- Client-side debugging or attach to the server.
- Support for Visual Studio 2010 and later

With this new .NET support, your Visual Studio IDE is used to create, edit, and debug SPs, UDFs, and triggers. A new API is provided along with Visual Studio templates for complete .NET integrated development. Compile on your client and deploy binaries to your server. Both C# and Visual Basic templates are provided.

A tutorial, which follows our standard FairCom tutorial model, is provided in:

<faircom>\drivers*tutorials**cSharp*



Visual Studio .NET Development

To aid in developing stored procedures (SP), user-defined functions (UDF), and triggers, FairCom provides a "c-tree Stored Procedure" extension to the Visual Studio IDE.

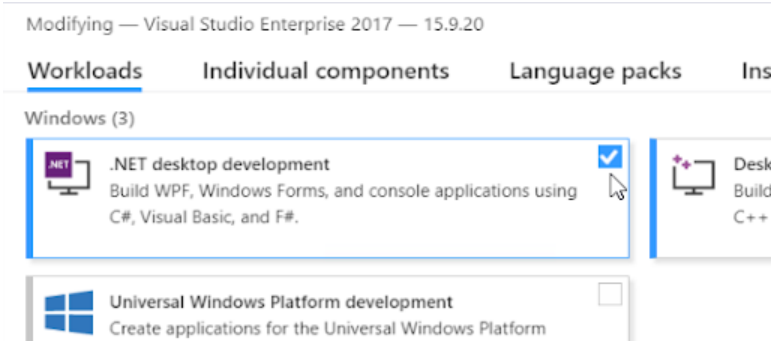
Note: C# .NET stored procedures are available on Microsoft Windows platforms only. If you wish to use stored procedures on other platforms, Java can be used for the stored procedures. See *Java Stored Procedures /readme/sql.stored.procs/*.

Note that the C# stored procedure tutorials that FairCom provides are not executables which can be launched from the command line, like most of the c-tree tutorials are. Instead, these tutorials must be installed into the database from within the Visual Studio IDE and then can be tested from within the Visual Studio IDE.

The .NET Framework 4.0, available from Microsoft, is required for writing SP, UDF, and triggers.

Prerequisite - Configure Microsoft Visual Studio properly

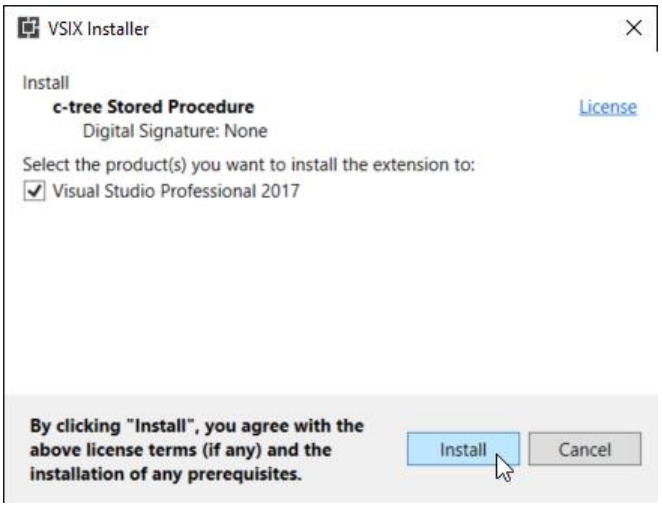
For Visual Studio 2017 to be able to build the tutorials, the ".NET desktop development" workload needs to be installed as part of Visual Studio. In Visual Studio 2017, click **Tools > Get Tools and Features**, which runs the Visual Studio installer. If the installer requires an update before it will run, allow that update. Find the ".NET desktop development" workload under the “Windows” group and make sure it is checked, as shown here:



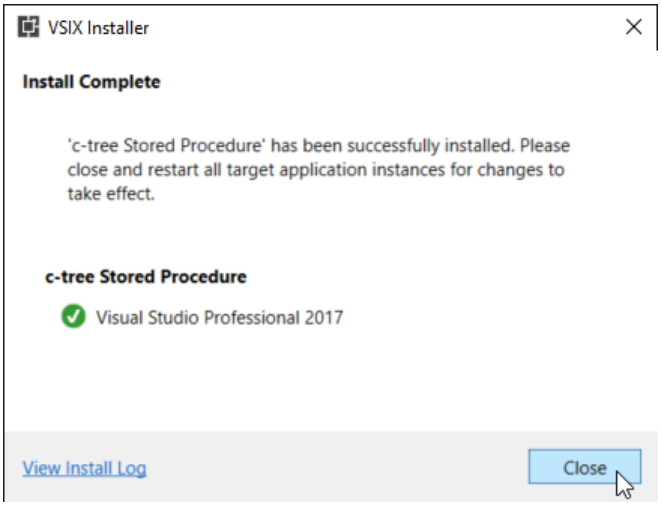
If it isn’t checked, give it a check mark and then press the **Modify** button to install that workload. This ensures that all of the tools needed to build and run these tutorials are installed.

c-tree Stored Procedure Extension Installation

FairCom distributes VSIX package installers, which install the c-tree Stored Procedure extension into Microsoft Visual Studio. These installers have been updated to support Visual Studio: 2017. Users with prior versions should uninstall their existing plug-in (following the instructions below) and install the new version for the latest compatibility. To install this extension, navigate to the *<faircom>\drivers\csharp.sql.storedprocs* folder with Windows Explorer, and then double-click the *.vsix* file that matches your version of Visual Studio. This will launch the VSIX Installer. Give it permission to run and make changes, and then click the **Install** button when you see a window like the following appear:



The install process might download and install one or more .NET Frameworks, so it's important to be connected to the internet while running this installer. Note that the install process can take several minutes. When the installation is complete, you should see a window like this:



This window is asking you to close Visual Studio (if it's open) and then restart it, in order for the changes to take effect. In other words, it is asking you to "reboot" Visual Studio.

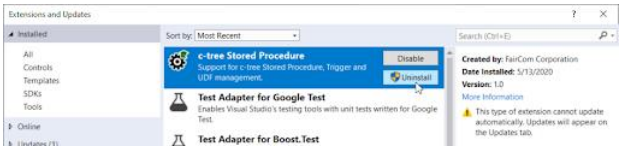
Note that this extension must be installed into Visual Studio before you load the solution file that contains the c-tree Stored Procedure tutorials.

Note also that this extension should be installed when Visual Studio is not running.

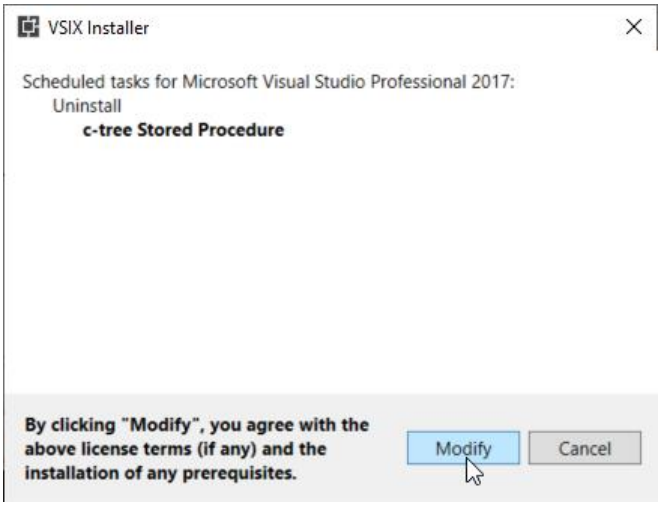
Extension Removal

Should you need to remove this extension, launch Visual Studio and do the following:

- **VS 2017** - Go to the **Visual Studio Tools** menu and select **Extensions and Updates**.
- Locate the “c-tree Stored Procedure” item template, and click **Uninstall**, as shown below. Note that the search tool is useful for finding the c-tree item.



- Clicking the **Uninstall** button will schedule the extension for un-installation. To make the un-install actually happen, exit Visual Studio. This will cause the VSIX Installer to run. Give it permission to run and then click the **Modify** button when asked.



Note that the un-install process can take several minutes.

Preparing to Write a .NET SP, UDF, or Trigger

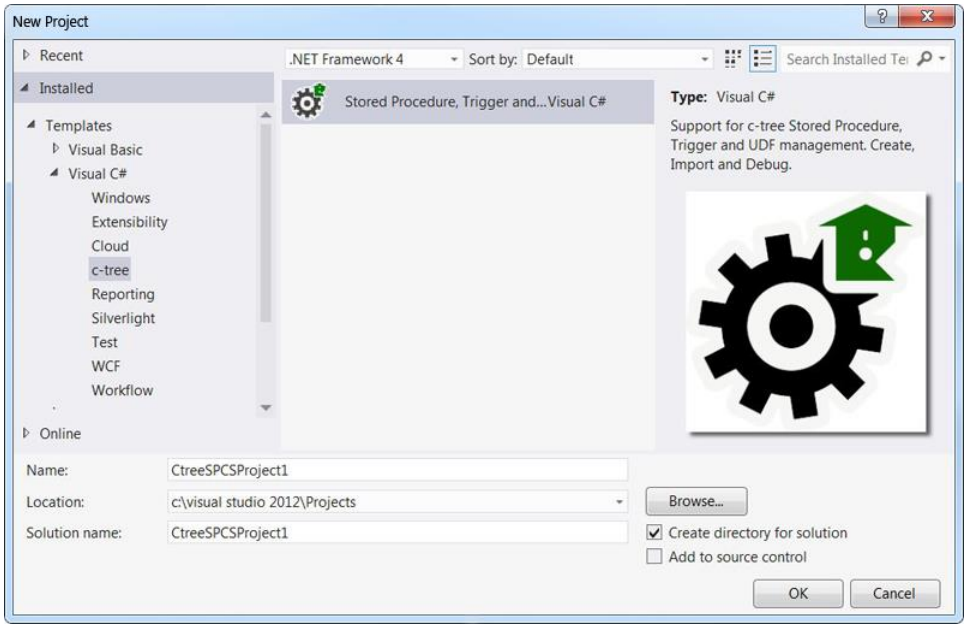
To write a .NET stored procedure (SP), user-defined function (UDF), or trigger, a new project must be created using one of the supplied templates (see *Visual Studio .NET Development* (https://docs.faircom.com/doc/*jspt/62766.htm*)). The steps below will create a C# stored procedure project.

Tutorials, which follow the format of the standard FairCom tutorials, are provided in *sdk*\<faircom>\drivers*tutorials*\.

Note: The extension for Visual Studio must be installed before you can perform the procedures in this section. See *Visual Studio .NET Development* (https://docs.faircom.com/doc/*jspt/62766.htm*).

1. Open Visual Studio.

2. Create a new project and, from the templates, select **Visual C#** > **c-tree** > **Stored Procedure, Trigger and UDF**.



3. Set the project name and location and click **OK**.
4. From the resulting dialog, pick the **Data Source** (pre-configured in Visual Studio) or **Custom** and provide the necessary server connection information (the server does not need to be running).

A new **Solution** and a new project will be created and opened. In Solution Explorer, you should see one folder for each type of project: Stored Procedures, Triggers, and UDFs.

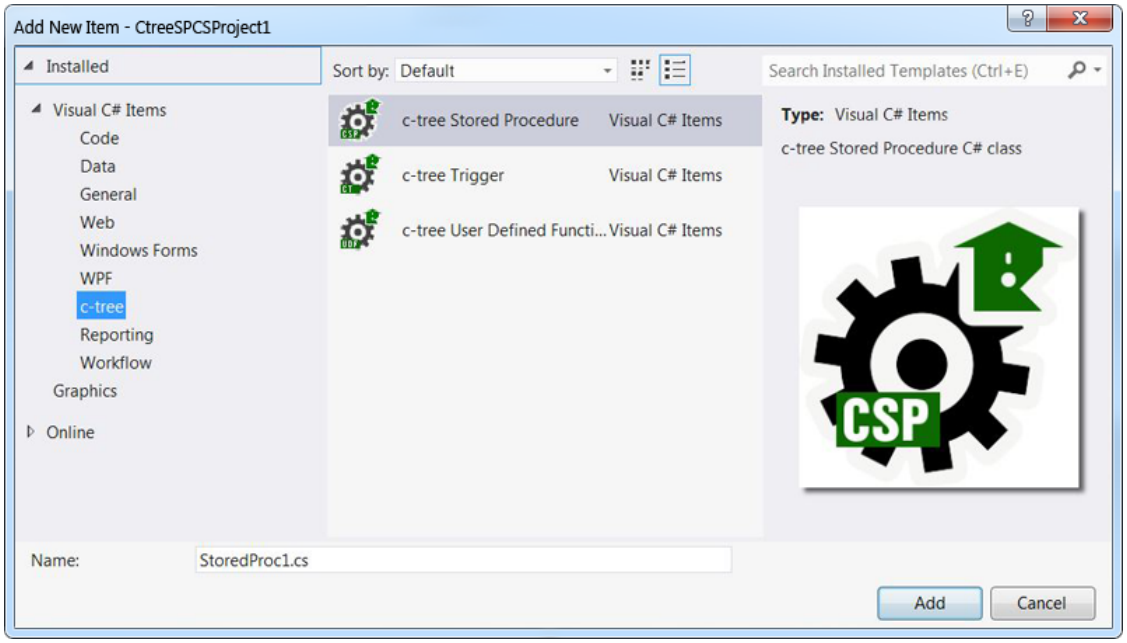
Writing a New SP, UDF, or Trigger

To write a .NET stored procedure (SP), user-defined function (UDF), or trigger, follow these steps after creating your solution and/or projects from the previous topic:

1. Right-click on your project (created in *Preparing to Write a .NET SP, UDF, or Trigger* (page 72)) or in any folder under your project in Solution Explorer and choose **Add/New Item**.
2. When the dialog appears, choose **Visual C# Items** > **c-tree** and choose the appropriate item.

   You may consider customizing the name of the *.cs* file that will be created. This name is not in any way associated with your stored procedure name.

3. When done, click **Add**.



4. A new dialog appears. Fill in the fields:
   - Specify your stored procedure name.
   - Adjust the namespace if you require or prefer a customized project. This namespace does affect execution or definition of your stored procedure; however, your choice may help organize your work.
   - Set the procedure owner, parameters, and result set.

   Click **OK**.

A new source file is created. The class you implement is already prepared and you implement the **ExecuteSP** method with your application logic, as described later.

Deploying the SP, UDF, or Trigger

Once coded, your SP, UDF, or trigger needs to be compiled and "deployed" to a server. Right-click it in Solution Explorer and choose **Deploy to Server**. One or more SP, UDF, or triggers can be deployed at the same time.

You are notified if any problems are encountered while the code is compiled and deployed to a c-treeACE SQL server.

Testing the SP, UDF, or Trigger

A stored procedure can be run to test its behavior by right-clicking in Solution Explorer and choosing **Test Procedure**. A window appears allowing you to set the stored procedure's parameters, execute it, and see the result generated.

Debugging the SP, UDF, or Trigger

By right-clicking in Solution Explorer and choosing **Debug Server** it is possible to debug a SP, UDF, or trigger while the server runs it.

The server must be active and reachable while debugging. For simplicity, the server should be running on the same machine, although it is possible to debug a remote server using Microsoft remote debugger.

You can set a breakpoint in the code, launch a debug server (not necessarily in this sequence), and execute your stored procedure via external tools such as c-treeACE SQL Explorer, your own program, or **Test Procedure** as described above.

Altering the SP, UDF, or Trigger

During development, it is often desirable to make code modifications and re-deploy a stored procedure, UDF, or trigger. This is supported by the FairCom extension for the Visual Studio IDE.

Importing Existing Code

It is possible to retrieve *existing* stored procedure source code located on the server but *not* in the project/solution. The **Import from Server** option is used to accomplish this. If the stored procedure was deployed with source code, this option retrieves that code and adds it to the current project to allow modification.

Altering Properties

By right-clicking on a stored procedure module in Solution Explorer and selecting **c-tree Properties** it is also possible to alter stored procedure parameters, result set, name and owner.

Writing the Code

The FairCom Visual Studio extension created a stored procedure skeleton for you. By default, the document created is shown with generated code automatically collapsed. Usually, you should not need to alter this code; however, for advanced customization, contact FairCom for additional customization details.

You write actual stored procedure code by implementing the **ExecuteSP** methods (see SqlStoredProc, SqlUdf, or SqlTrigger in *The .NET Ctree.SqlSP Assembly* (https://docs.faircom.com/doc/*jspt/62769.htm*)).
In writing your code, you can add new methods if necessary.

Note: You should not change the prototype of the **ExecuteSP** methods, which reflect actual stored procedure arguments.

Returning Errors

A stored procedure throws an exception to generate an error sent to the client. An end user always sees a c-treeACE SQL error **-20142** returned in this case. The error message indicates an error in executing the stored procedure followed by the error number and an associated message, such as:

```
error(-20142): Error in Stored Procedure/function Execution - error(-34567): my error message.
```

The API used to write stored procedures and the .NET runtime may also throw exceptions. If these are not caught by your exception handling, they are passed to the c-treeACE SQL runtime and result in a **-20142** error.

Null Handling

For easier null value handling, SQL types are mapped into nullable .NET types. Whenever a “SQL value” is referenced, the type used is nullable. Whenever an object representing a SQL value is set to null, it indicates a “SQL value” was null (reading a column/parameter value) or it is set to null when updating (setting a column/parameter value).

Returning a Result Set from a Stored Procedure

A stored procedure may return a result set. In this case you must add columns to the result set row and then add the row to a result set.

This is accomplished as shown below:

1. Create an instance of the SqlSpResulsetRow class by invoking the **NewResultSetRow()** method.
   ```
   SqlSpRow RSrow = NewResultSetRow();
   ```
2. Set column values in a result set using the column index. For easier code writing, the generated code contains constants with the name of the result set columns and the values are the column indexes.
   ```
   RSrow[RSCol.mycolumn].Value = "";
   ```
3. Add the row using **ResultSetAddRow(***SqlSpRow RSrow***)**. Notice that this method creates a copy of the row, so the SqlSpRow object can be reused for additional rows.

A stored procedure may also have OUT parameters that must be set before returning. This is enforced by the C# language as the **ExecuteSP** prototype implements “out” for OUT parameters.

A stored procedure may also have INOUT parameters that may or may not be set before returning. In this case the **ExecuteSP** prototype implements “ref” for INOUT parameters.

Returning a UDF Result

A user-defined function is declared with a “return” value that is the return value of the **ExecuteSP** method.

Accessing Old and New Rows in Triggers

Triggers do not have IN, INOUT, or OUT parameters but may receive one or two rows depending on the definitions of REFERENCING, OLDROW, and NEWROW.

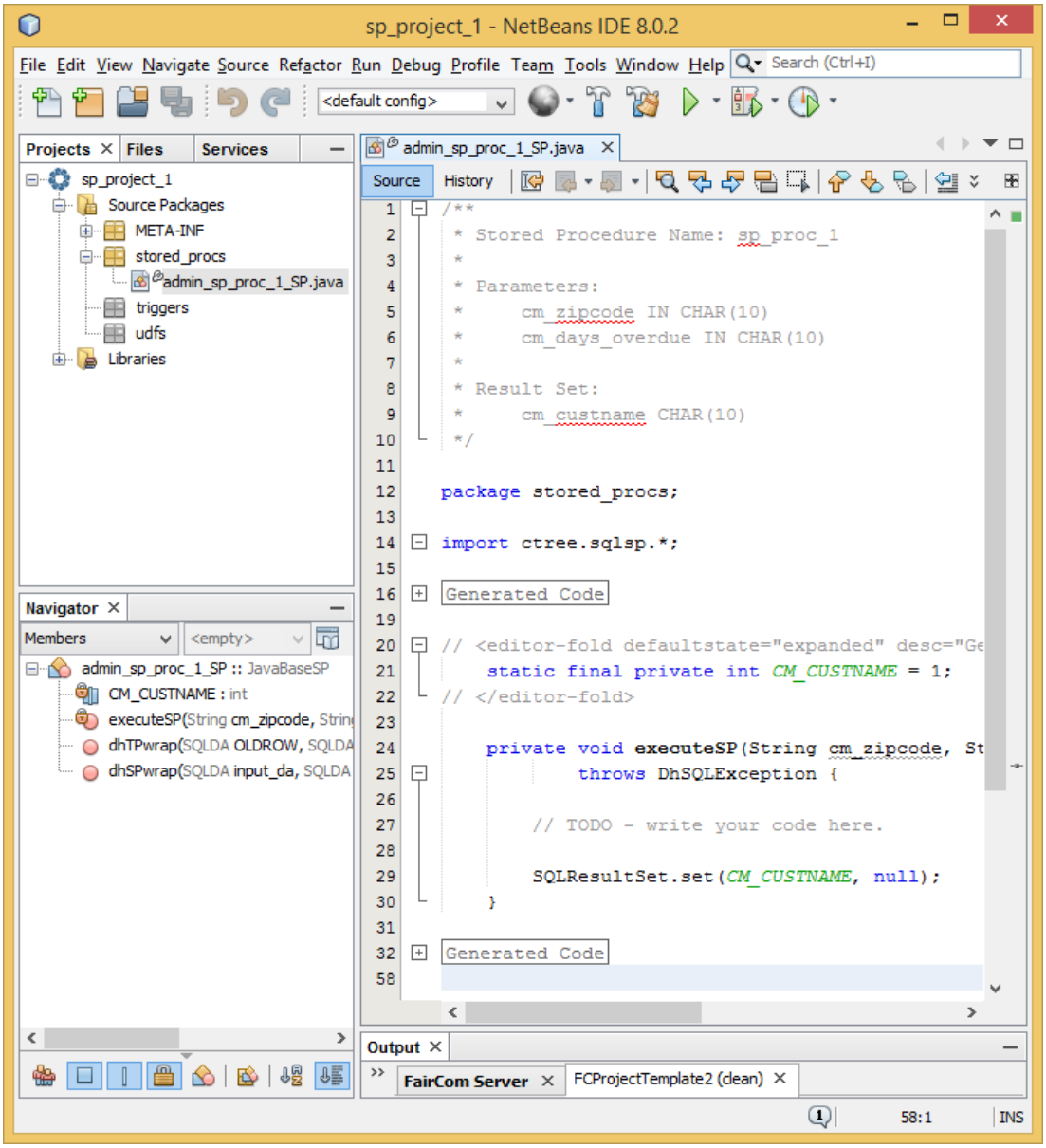
These rows are passed, when referenced, as SqlSpRow objects to the **ExecuteSP** method.

The methods of the SqlSpRow class can then be used to retrieve the definition and values of columns. As is the case with stored procedure result set columns, the generated code contains constants with the names of the result set columns and the values of the column index.

7.2 NetBeans Plugin for Java Stored Procedure Development

FairCom DB includes a Java NetBeans plug-in for advanced development potential. Take advantage of the complete NetBeans IDE in creating and maintaining new and existing Java stored procedures. Existing Java stored procedure investments can continue to be used with much easier maintenance and deployment capabilities.

NetBeans is a complete and easy-to-use IDE for Java developers, providing a modern natural framework for Java stored procedure development. FairCom now provides a NetBeans plug-in enhancing stored procedure development and testing. This includes expected productivity features such as syntax checking and in-line debugging.



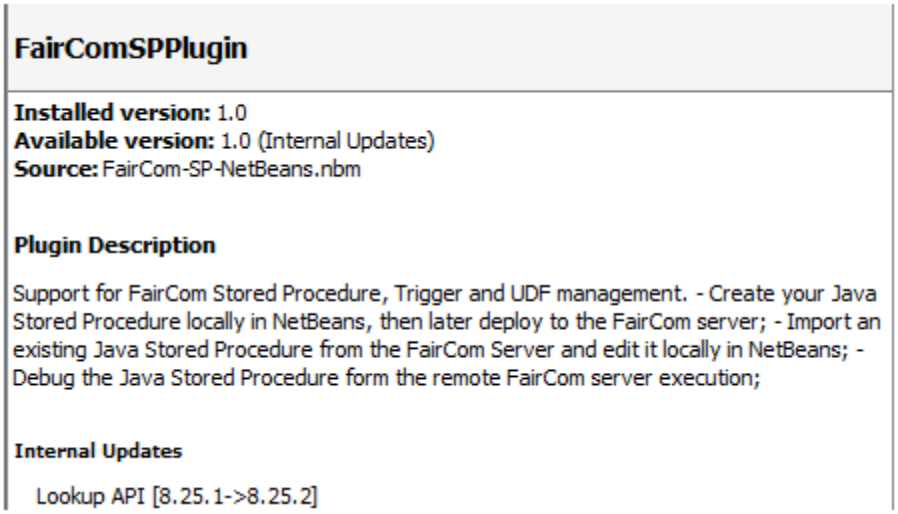
Our NetBeans Stored Procedure plug-in allows developers to compile and test Java procedure code via a client-side interface and deploy final procedure code to the server. It is not necessary to provide a full JDK on the server as development can now be centralized at the developer workstation. Debugging within the IDE is possible by retrieving procedure source code from the server. In addition, existing procedure code can easily be brought into this framework greatly extending maintenance of your current deployed procedures.

For details, see **NetBeans Plugin for Java** (, **/doc/jspt/62755.htm**) in *FairCom Java & .NET Stored Procedures*.

Installing the NetBeans Plugin

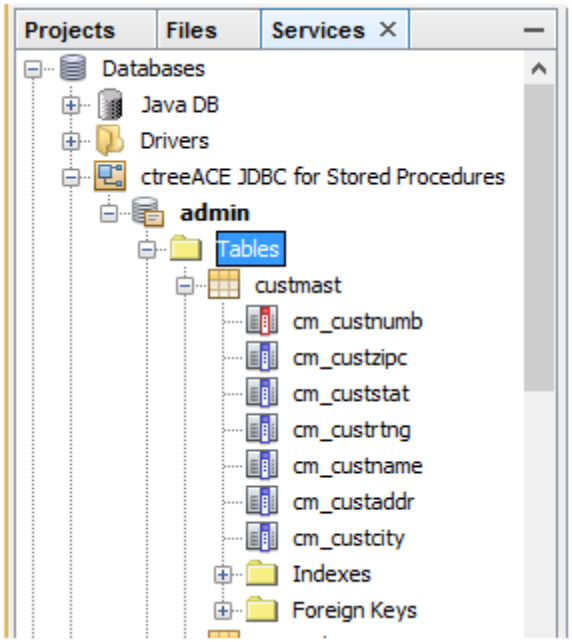
To install the plugin in NetBeans:

1. From the **Tools** menu, select **Plugins**. On the **Downloaded** tab click **Add Plugins**.



2. Look for *FairCom-SP-NetBeans.nbm* located in:
   \FairCom\V10.4.0\win32\sdk\sql.stored.procs\netbeans
   Select it and click **Open**.
   You will see two *.NBN* files. Install both.

3. Click **Install**.

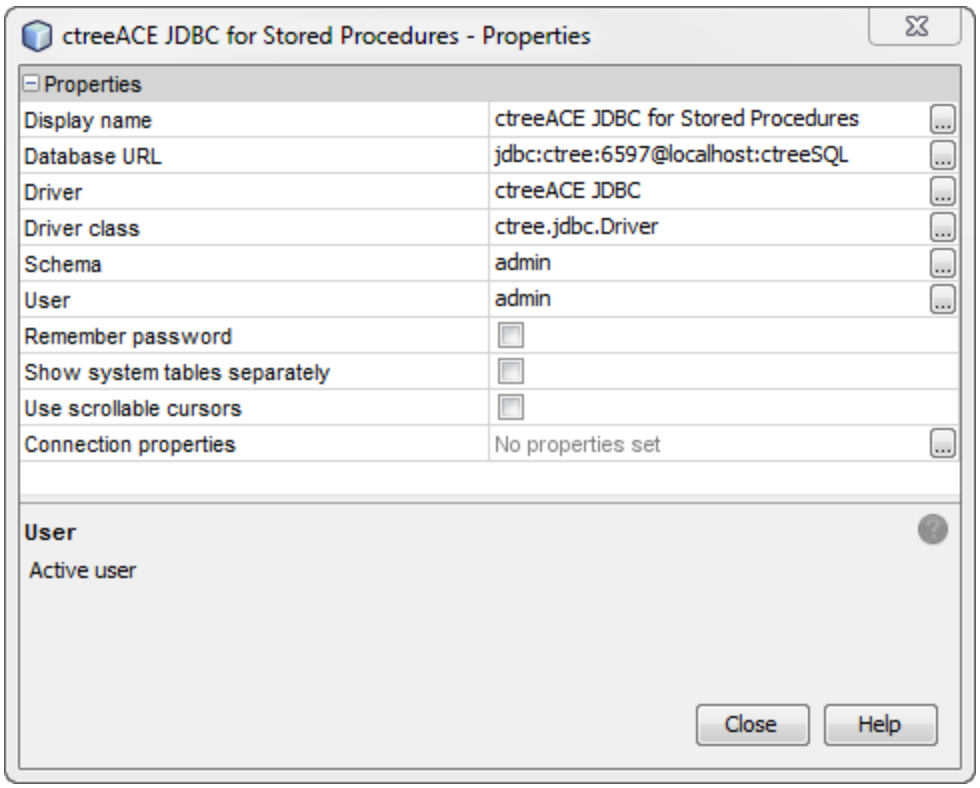


Setting up the Connection to the Server

To set up the connection to the server:

1. In NetBeans, click the **Window** menu and select **Services**.
2. Open the **Databases** branch. Create an entry called **ctreeACE JDBC for Stored Procedures**, which should use the **c-treeJDBC** driver.

3. Right-click on **ctreeACE JDBC for Stored Procedures** and set its properties as shown below:



Verify the "Database URL" (also commonly known in SQL terms as the connection string). The format is as follows:

```
jdbc:ctree:port@host_name:db_name
```

The string is made of the following components:

- *jdbc:ctree* - An identifying protocol and subprotocol string for the c-treeACE SQL JDBC Driver.
- *:port* - The port number associated with the c-treeACE SQL server on the host system (e.g., `6597`).
- *@host* - The name of the server system where the database resides (`@localhost` is default)
- *:dbname* - This is the c-treeACE SQL database name (see `SQL_DATABASE` in ***ctsrvr.cfg***) or any applicable database registered with the c-treeACE SQL database, such as `ctreeSQL`.

The default will be:

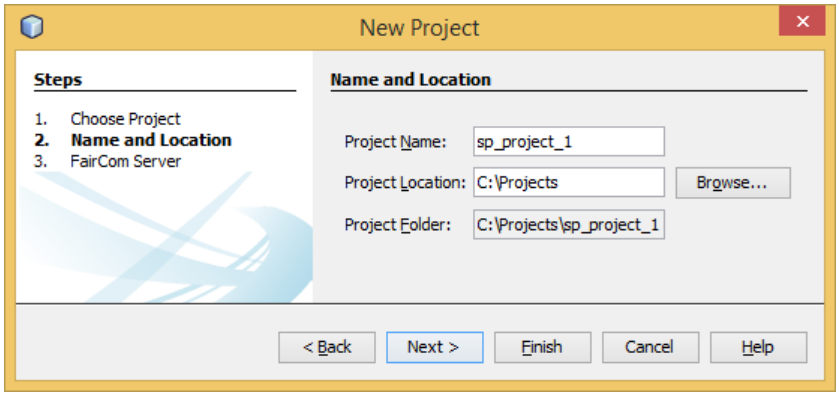
```
jdbc:ctree:6597@localhost:ctreeSQL
```

4. Double-click on **ctreeACE JDBC for Stored Procedures**. The icon should become "unbroken."

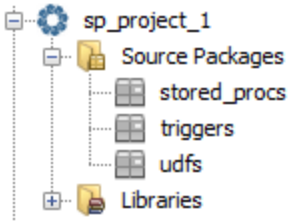
Editing Stored Procedures

To edit stored procedures in NetBeans:

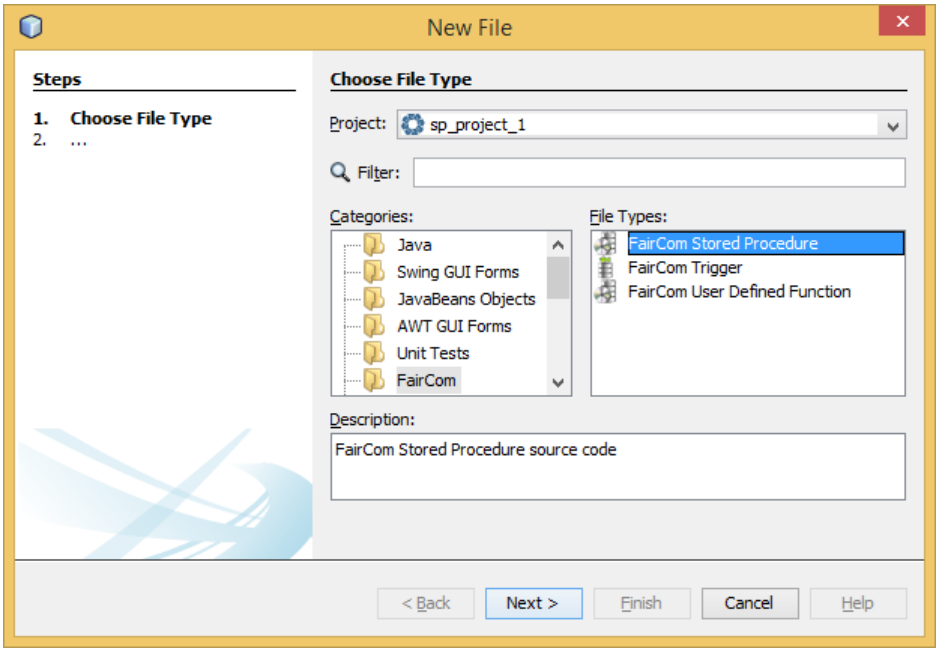
1. Create a new project using **File > New Project**.
2. In **Categories**, select **FairCom** and in **Projects** select **Stored Procedures & ...** then click **Next**.
   Update the project name and location information as you desire then click **Next**.
3. Select the desired JDBC connection for the project to be able to deploy and retrieve stored procedures. This selection can be changed later in the project properties under **FairCom Server**.



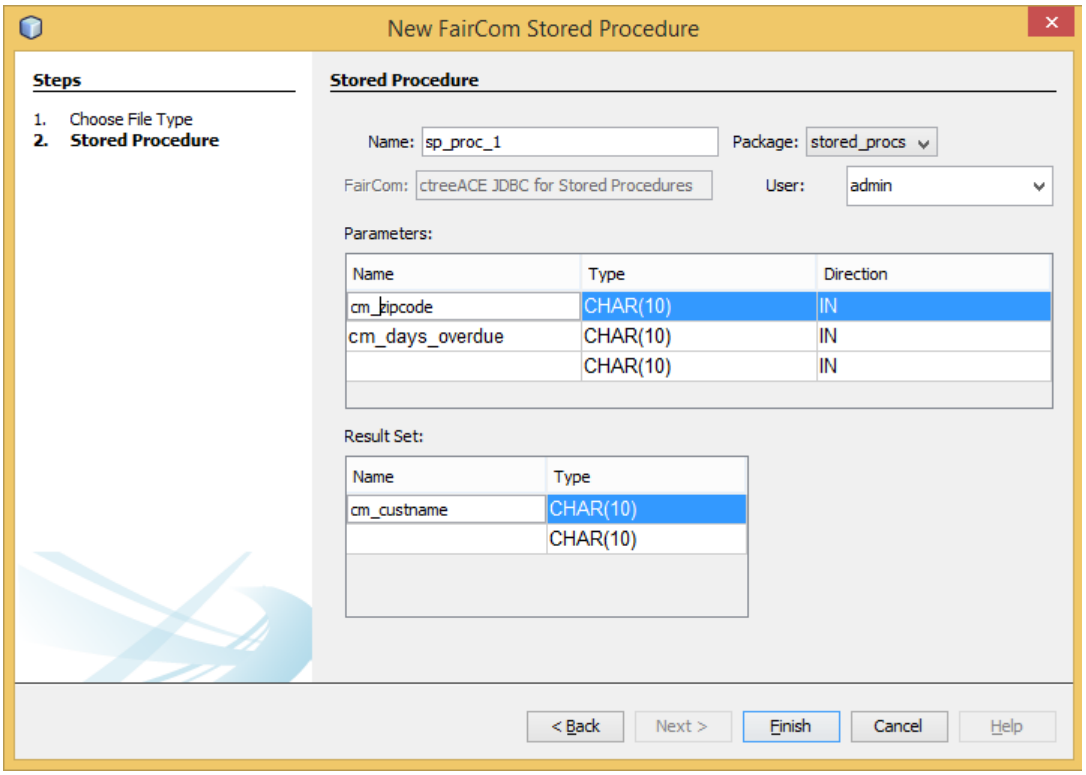
4. Click **Finish**.



5. Create a new stored procedure using these steps:
   a. Right-click the project and select **New** then select **Other** (if the project is not already listed).



a. Click **Categories > FairCom > File types** and select the appropriate type for the project you desire to build (e.g., FairCom Stored Procedure). Follow the wizard for additional steps.



a. Write the code for your stored procedure.

> **Note**: Stored procedure names are case-sensitive (as is any other database object). To reference a case, you need to quote it as is done for other identifiers. For example:
> `CALL "Tutorial1"()`

1. Import the project from the server by right-clicking on the project *stored_proc* package and clicking **Import** from server.

Enabling Debugging

To debug a stored procedure, add the following keywords to *ctsrvr.cfg*. Be sure to restart the c-treeACE Database Server so these changes will take effect.

```
;JAVA DEBUG
SETENV DEBUG_JVM=S
SETENV DEBUG_JVM_PORT=45987
```

7.3 c-treeACE SQL Deployment Utilities for Stored Procedures, UDF, and Triggers

In V11 and later, c-treeACE SQL stored procedure dump and deploy utilities render it unnecessary to require both a Java compiler (JDK) and run-time engine (JRE) on every server. A single utility is now provided allowing deployments to dump stored procedures (SP), user defined functions (UDF), and triggers created on one c-treeACE SQL installation. A second utility deploys procedures, etc. to other remote c-treeACE SQL installations.

- A JDK is still required to develop and compile SPs, UDFs, and triggers on one server, but is not required on remote deployed servers.
- A JRE is still required on each deployed c-treeACE SQL installation that will execute SPs, UDFs, and triggers.

dbschema

After developing a SP, UDF, or trigger, call the **dbschema** utility with the new *-b* option to dump your procedure routines such that they can be deployed with the new **dbdeploy** utility.

Usage

```
dbschema [-d] [-b] [-u username] [-a authentication] [-o outputfile]
         [-t tablenamelist] [-p procedure_name_list]
         [-f function_name_list] [-T trigger_name_list]
         [dbsegment_name]
```

- *-d* - Dump the data
- *-b* - Dump stored procedures, user-defined functions, or triggers in binary format so they can be deployed using **dbdeploy**
- *-o* - Output file (default: screen)

dbdeploy

The SP, UDF, or trigger in the output file generated using **dbschema** *-b* can be deployed to other c-treeACE SQL installations with the **dbdeploy** utility.

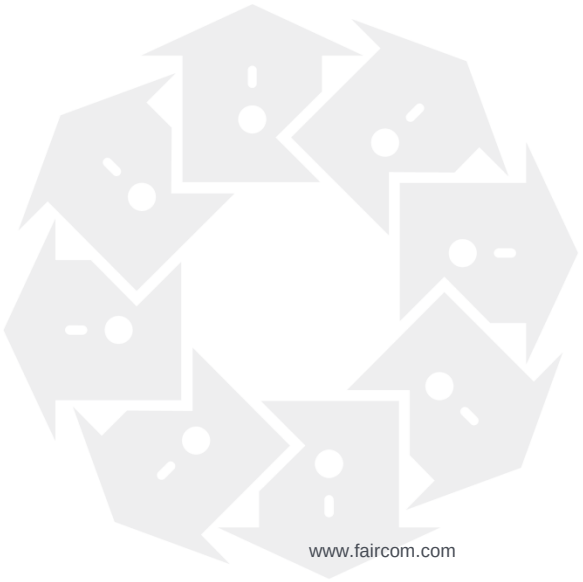
Usage

```
dbdeploy [-u username] [-a authentication] [-i inputfile]
         [dbsegment_name]
```

- *-u* - Username identifiable to the DBMS
- *-a* - Password for authentication
- *-i* - Input file.

8. User-Defined Partitioned File Conditional Expressions

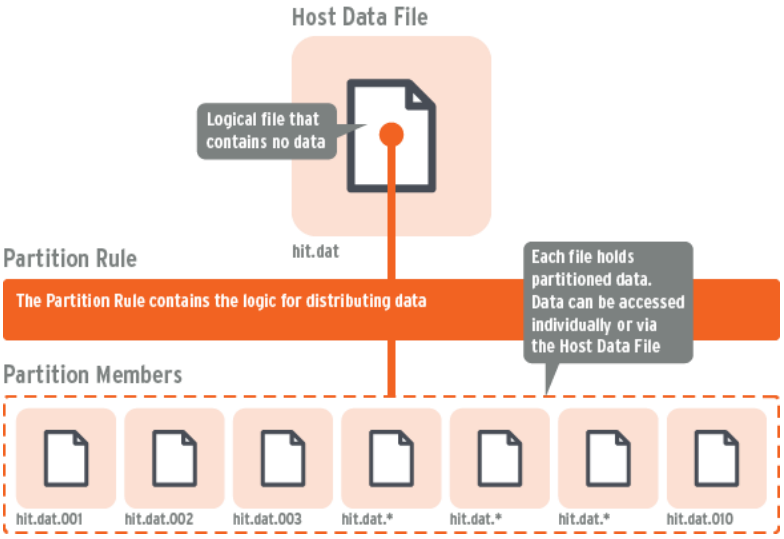
Partitioned files become much easier to implement with complete new support for user defined conditional expressions. Easily add this powerful file partitioning feature for fast and efficient bulk data management.



8.1 User-Defined Conditional Expressions for Easy Partitioned File Creation

Partition files are easily created with user-defined conditional expressions. Partitioned tables are fully supported via both SQL and c-treeACE ISAM applications.

A partitioned file logically appears to be one data file (and its associated index files). It is actually a set of files whose contents are physically partitioned by the value of a *partition key*.



Partitioned files are intended for applications requiring fast purging and archiving of large amounts of data at once. As both data and index files are partitioned, this permits data within a defined range of partition key values to be rapidly purged or archived (rather than deleting each record within this range).

Consider rapidly acquired log data. Frequently, this data has a lifespan of days to months. After this time it is usually purged or archived for permanent storage. This can take time with large amounts of new data that have since been acquired. By avoiding lengthy key searches for expired data over the entire data set, it is much better performing to have all related data in a single silo and operate on it with a single operation. Partitioned files give you this powerful ability.

For background information regarding partitioned files, refer to *Partitioned Files* (https://docs.faircom.com/doc/*ctreeplus/partitioned-files.htm*) in the *c-treeACE Programmer's Reference and Function Reference Guide*.

8.2 Conditional Expressions and Partition Rules

Conditional expressions make it easy to build a partition rule. A partition rule is a conditional expression ultimately evaluating to a number. That number associates data with a specific partition within the file.

c-treeACE expression parsing can now evaluate an expression into a numeric representation to be used for the partition rule. Many built-in conditional expression functions exist for flexible rule generation. Time and date based functions, numeric functions, and string manipulation functions are all available. c-treeACE expression syntax can even reference complex functions via an external shared library (DLL) in calculating partition numbers.

If the table has an embedded *DODA* resource and a conditional expression references schema segments, an expression can refer to index key fields explicitly by *DODA* name. Together with field names, complex expression rules can be crafted.

Prior to c-treeACE V11, partitioned files required these rules to be hard-coded at compile time (specified in *ctpart.c*) and did not allow run-time flexibility in adapting rules for unique deployed environments.

By combining existing c-treeACE advanced support for conditional expressions with our high performing partitioned files, applications can now create exact expression rules directly when creating partitioned files. In addition, existing partitioned files can have their partition rule changed and subsequently rebuilt based on a new rule logic allowing applications to grow and adapt with their customer needs.

Partitioned files are supported directly from the c-treeACE ISAM API, FairCom DB API and c-treeACE SQL.

Performance

Additional processing of advanced expression parsing can potentially impact performance to a small degree. Should performance become an issue, you might consider an external DLL to efficiently evaluate partition rules directly.

8.3 Partitioned Files in c-treeACE SQL

In c-treeACE SQL, a table is partitioned when a partition index is created. To create a partition index with conditional expressions for partitioning, use the following syntax:

```
CTREATE INDEX .... STORAGE_ATTRIBUTES 'partition=<expression>'
```

To create a partition index forcing hard-coded rules (pre-V11), the following is supported:

```
CTREATE INDEX .... STORAGE_ATTRIBUTES 'partition'
```

To change a partition rule on an existing partitioned file, call ALTER INDEX with your new rule in the STORAGE_ATTRIBUTES clause.

Example

This script demonstrates partition file rules in SQL. It creates a table, `prtest`, with an index on the integer field `f7`. The `storage_attributes` clause define a partition rule in which each record is stored in a partition number equal to the value of field `f7` plus `1`:

```
create table prtest (f1 integer, f2 char(50), f3 char (50), f4 char(50), f5 timestamp, f6 varchar(50), f7
integer, f8 time);
create index pridx on prtest (f7) storage_attributes 'partition=f7+1';
```

With a rich assortment of conditional expression functions available, much more complex rules can be created.

For example, a table containing a field "invoice_date" and requiring monthly partitions can be created with this simple expression:

```
month(invoice_date)
```

For syntax details, refer to *Conditional Expression Parser* (https://docs.faircom.com/doc/*ctreeplus/25930.htm*) topics in the *c-treeACE Programmer's Reference and Function Reference Guide* (https://docs.faircom.com/doc/*ctreeplus/*).

Example

Using functions to convert Unix time_t fields to c-tree Date and Time types (TIMET2CTDATE) in a partitioned file expression to partition into months since Jan, 2010:

```
CREATE TABLE unixtest (name CHAR(10), u_date INTEGER)
    or
CREATE TABLE unixtest (name CHAR(10), u_date BIGINT)

CREATE INDEX unixtest_date_idx ON unixtest (u_date) STORAGE_ATTRIBUTES 'partition=( ( YEAR( TIMET2CTDATE(
u_date) ) -2010) * 12) + MONTH ( TIMET2CTDATE(u_date))'

    Date String                              Unix Date       Partition created
    Mon, 23 Feb 2015 11:01:32 GMT    1424689292      62
    Sat, 23 Jan 2016 11:01:32 GMT    1453546892      73
    Tue, 23 Feb 2016 11:01:32 GMT    1456225292      74
    Wed, 23 Mar 2016 11:01:32 GMT    1458730892      75
```

For syntax details, for the TIMET2* functions, see *C Language Equivalents*.

8.4 FairCom DB API Partition File API Support

Partitioned file support is extended to the FairCom DB API API. While creating a file it is possible to call **ctdbSetTablePartitionIndexNbr** to set the partition index, **ctdbSetTablePartitionNumberBits** to set the number of bits reserved for partition numbers, and **ctdbSetTablePartitionRule** to set the partition rule.

On existing tables, after calling the above function you then call **ctdbAlterTable** forcing an *CTDB_ALTER_FULL* action.

ctdbSetTablePartitionRule

This FairCom DB API function sets partition rules:

```
ctdbEXPORT CTDBRET ctdbDECL ctdbSetTablePartitionRule(CTHANDLE Handle, pTEXT expr);
```

- *expr* - The expression, *expr*, will be evaluated against the key for the partition index. It must evaluate to an integer.

  The partition rule uses standard c-tree expression syntax.

8.5 c-treeACE ISAM Usage

Partitioned files are created with standard c-tree APIs. A partitioning conditional expression is defined with the **PTADMIN** Partition Administration API call and the *ptADMINrule* parameter. Specify an extended (*Xtd8*) *ctPARTAUTO* mode in the *x8mode* parameter of an extended file creation block. While partitioned files require an extended (*Xtd8*) header they do not have to be *HUGE* files unless the logical file size will exceed the 2GB/4GB limit.

A partition key is set when the file is created using the *prtkey* parameter of the extended file creation block. The default is 0 (the first key associated with the data file). Set this value to the relative key number for the desired index if the default is not appropriate for your application.

A partition file name is the base data file name with a 3-digit raw partition number as the file extension. Only automatic naming is available in this mode.

Note: Alternative file naming is possible with custom modifications to the *ctpart.c* module and recompiling your FairCom Database Engine.

By default, 16 bits of the 64-bit record offset reference the raw partition number, allowing each partitioned file to support up to 65,535 member files over its lifetime, thus also somewhat limiting overall file size (i.e., the more bits used for partition numbering, the smaller the overall size of the file can be). The maximum value is 32-bits (4,294,967,295 member files). This numbering vs. size tradeoff is set at file create time using the *callparm* parameter of the extended file creation block (*Xtd8*), where a value of 0 defaults to 16-bits. Values less than 4-bits default to 4-bits (maximum 15 member files).

Note: Default partition naming, partition rules, and maximum number of partitions are used by default when not defined by the *Xtd8* extended parameter block and the **PTADMIN** API call.

8.6 Managing Partitions

Each c-treeACE data partition and index pair resides as an independent data set. Their purpose is for ease and speed of data management. They can easily be purged, archived, re-activated, and even rebuilt as individual data silos with a single operation. Administration is done through the c-treeACE **PTADMIN** API function.

SQL Administration

c-treeACE SQL includes built-in procedures to administer partitions directly from your SQL application. The **fc_ptadmin_num() /doc/sqlops/57673.htm** procedure provides access to many basic administration functions.

```
call  fc_ptadmin_num('admin',  'custmast',  'archive',  123)
```

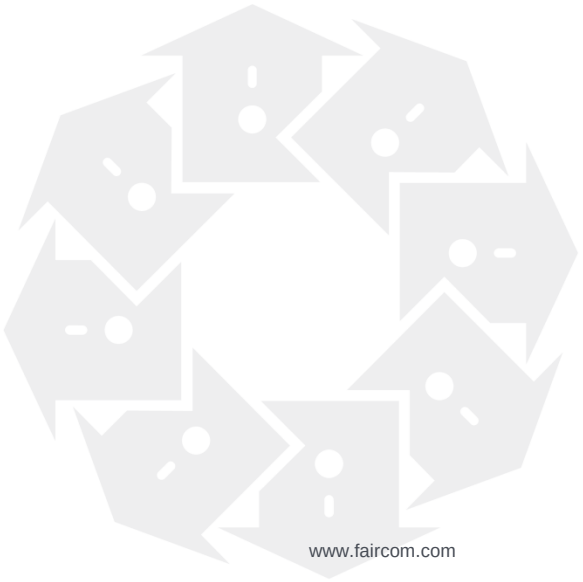
To identify current lowest and highest active partition numbers, the built-in SQL procedure **fc_get_partbounds()** returns this information.

```
call  fc_get_partbounds('admin',  'custmast')
```

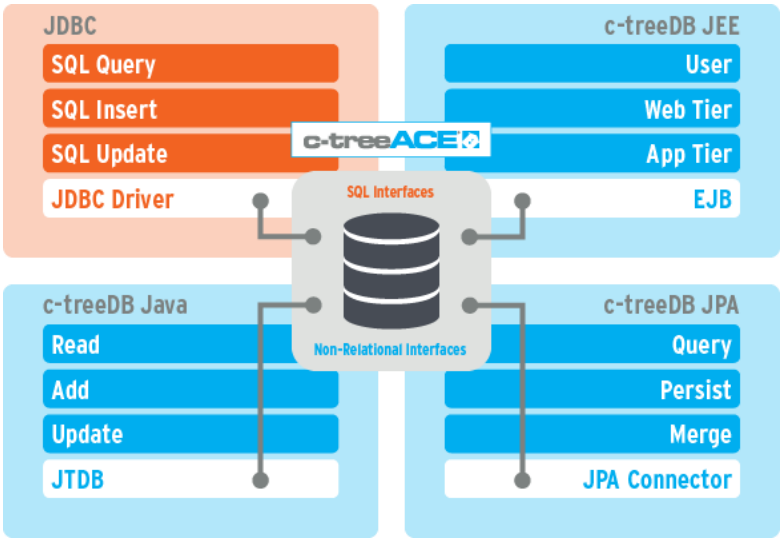
9. c-treeACE Java Edition Solutions

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

With this release, c-treeACE places a new focus on Java. We have expanded our support for Java offering both relational SQL and navigational NoSQL interfaces.



9.1 Everything for Java



FairCom continues its dedication to cross-platform support and Java remains one of the strongest global standards in this regard. Java retains its position as the choice of distributed enterprise software development. V11 expands c-treeACE Java offerings (page 90), with multiple Java technologies for desktop to distributed enterprise solutions.

- FairCom DB API Enterprise Java Beans: Integrate c-treeACE as a Resource Adapter
- FairCom DB API JPA: NoSQL Java Persistence API quickly converts existing applications
- c-treeACE JDBC: Type IV JDBC driver for industry-standard applications
- FairCom DB API Java: Fast, NoSQL access to c-tree files using FairCom DB API for Java

c-treeACE includes everything Java programmers require:

FairCom DB API JPA (page 92) - ISAM methods implemented under a new Java Persistence API (JPA), allowing c-treeACE usage from Persistence Java applications. Replacing SQL-based Hibernate, in targeted instances, our NoSQL-based JPA has proven much faster than standard SQL JDBC. Of course, we also support Hibernate as pure SQL via c-treeACE JDBC.

The Java Persistence API (JPA) provides a 100% object-oriented abstract layer to manage relational entity data without specific data I/O commands, which allows applications to be independent of platform and database. Unlike many SQL-based JPA implementations, such as Hibernate, FairCom uses our low-overhead NoSQL key-value interface technology for improved performance. The FairCom JPA can replace Hibernate and existing RDBMS from applications with minimal, if any, application code changes.

FairCom DB API Enterprise Java Beans (page 94) - Enterprise Java Beans make ISAM methods available for Java applications deployed on an application server framework. For enterprise class, distributed applications, integrating multiple environments, such as mainframes and open systems that require high performance ISAM access to c tree data.

The FairCom Java Enterprise Edition (JEE) offers an implementation of a Resource Adapter for JEE Application Servers such as Glassfish. It publishes c-treeACE NoSQL methods, which allows

enterprise-class, distributed applications to view c-treeACE data as a registered resource. This API is well suited for bringing high-performance data access to multi-tier java applications through the c-treeACE NoSQL layer.

c-treeACE JDBC (page 94) - Standard SQL access to c-tree ISAM files. For traditional Java applications (POJO), accessing c-tree data as any other SQL-oriented database can immediately take advantage of the c-treeACE SQL type IV JDBC driver.

The Java Database Connectivity (JDBC) API, the industry standard for database-independent connectivity, provides a standard connection between Java and the c-treeACE SQL database engine. c-treeACE JDBC provides an API that allows you to exploit Java's "Write Once, Run Anywhere" capabilities for applications that run on different platforms and access enterprise data.

FairCom DB API Java (page 94) - FairCom DB API Java provides ISAM access methods to c-tree ISAM files, available for Java applications. For Java applications that need to access c-tree data via ISAM methods with high throughput, performance, and low-level control.

c-treeACE NoSQL Java Interface Technology, commonly referred to as FairCom's c-treeDB Java API (or JTDB for short), provides Java developers a unique record-oriented, non-SQL, Java framework to manage data with the c-treeACE NoSQL database engine. This interface offers the performance advantages of direct access to records while still allowing full Java access to the same data available through the industry-standard JDBC interface. JTDB is perfect for single-tier Java applications.

And don't forget, c-treeACE SQL supports stored procedure, triggers and user defined functions development in Java. Moving your business logic server side not only improves performance, it also enforces business rules and allows for controlled updates as rules are maintained over time. V11 brings a new Netbeans IDE environment (, /doc/jspt/62755.htm) to your Java stored procedure development platform.

9.2 Java Persistence API (JPA) with c-treeACE

The Java Persistence API (JPA) is defined as part of the EJB 3.0 specification for accessing, persisting, and managing data between Java objects/classes and a relational database. JPA is the standard approach for Object to Relational Mapping (ORM) in Java.

JPA is a specification for a set of interfaces; open-source and commercial JPA implementations are available. JPA requires a database to hold the persisted objects. Most, if not all, Java EE 5 application servers should support JPA.

JPA allows POJO (Plain Old Java Objects) to be persisted without implementing any interfaces or methods (as was required by the EJB 2 CMP specification). Using JPA, the object-relational mappings for an object (defining how the Java class maps to a relational database table) can be defined through standard annotations or XML. JPA defines a runtime EntityManager API, which processes queries and transactions. JPA provides an object-level query language, JPQL for querying the database.

EclipseLink is the JPA implementation used with the FairCom Server. It is a reference implementation of the JPA 2.1 specification which has been extended to support non-relational (NoSQL) databases.

Java persistence consists of four areas:

- The Java Persistence API
- The query language
- The Java Persistence Criteria API
- Object/relational mapping metadata

Common JPA implementations, such as Hibernate, map data interactions to relational SQL commands with a RDBMS to persist data and is fully supported with c-treeACE SQL and JDBC.

FairCom DB API JPA is a novel new navigational (NoSQL) implementation: Instead of mapping to relational SQL, we map to our ISAM interface, benefiting from reduced resource usage and improved performance.

Current JPA Java applications can easily switch to the FairCom's FairCom DB API JPA and take advantage of our ISAM persistence implementation without application changes. Remove additional unneeded SQL depending on the type of application.

FairCom's FairCom DB API JPA replaces Hibernate and your RDBMS from any application that has been developed using this architecture.

Benefits

The advantage of using JPA over other Java persistence technologies is that it is very easy to set up and use. Other APIs require you to implement or extend superclasses or interfaces or to write huge and clumsy XML files. With JPA you only need to annotate a model class (business object) with **@Entity** and you need an ID field that you annotate with **@Id**.

You can tell JPA to store instances of those annotated classes. You can load them using the ID value or using a query language similar to SQL called JPQL.

What requires 100 lines of code in other Java persistence technologies takes only 10 when you use JPA.

Implementation

The JPA implementation used with the FairCom Server is EclipseLink, the reference implementation of the JPA 2.1 specification. In 2012 it began an effort to support NoSQL databases. Support for a new NoSQL database is added by writing a plugin that conforms to the JCA and CCI APIs. FairCom has written such a plugin for EclipseLink based on its JTDB Java API.

You will need **EclipseLink version 2.6.1** and the **c-treeACE NoSQL** plugin. Both can be found in the c-treeACE package.

Limitations

The NoSQL JPA plugin for EclipseLink uses the c-treeACE JTDB API, which is currently not platform-independent. The plugin needs one DLL or .so file in the system path. The c-treeACE

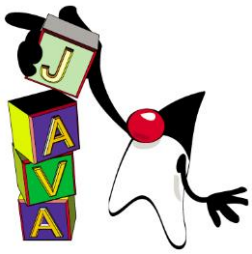
dialect for *Hibernate* does not have this limitation because it uses the c-treeACE JDBC driver and the c-treeACE SQL Server. The c-treeACE JDBC driver does not rely on a DLL or .so file to function, which makes it a more platform-independent solution. However benchmarks have shown the NoSQL approach to be significantly faster because it cuts through all the SQL layers.

Similarly if you use the JDBC API directly you also do not rely on any DLL or .so files.

In summary, if your platform allows binary library files like DLL or .so files, FairCom recommends using the NoSQL JPA variant.

9.3 Enterprise Java Beans

FairCom offers an implementation of a Resource Adapter for JEE Application Servers such as Glassfish. This JEE adapter publishes c-treeACE ISAM methods making them available for applications using Enterprise Java Beans (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.



9.4 c-treeACE JDBC

Java Database Connectivity (JDBC) API, the industry standard for database-independent connectivity, provides a standard connection between Java and the c-treeACE SQL Database Engine. JDBC provides an API for c-treeACE SQL-based database access, which allows you to exploit Java's "Write Once, Run Anywhere" capabilities for applications that require access to enterprise data. An application written in Java alleviates the challenge of writing different applications to run on different platforms.

The c-treeACE JDBC driver is a full Type 4 (native Java) JDBC driver, and as such, is platform-independent. This file can be located anywhere in an application installation and is referenced solely with an appropriate Java CLASSPATH specification. The c-treeACE JDBC driver requires no other changes to other system components.

9.5 FairCom DB API Java

c-treeACE Java Language DataBase Interface Technology, commonly referred to as FairCom DB API Java API, provides Java developers a unique record-oriented, non-SQL, Java framework to manage data with the c-treeACE database engine. This interface offers alternatives to the Java developer requiring direct access to data records for performance advantages while still allowing full JDBC SQL access to the same data.

The FairCom DB API Java API provides libraries that allow Java programs to access the c tree database core. This approach provides application developers with a simple interface into the powerful c-treeACE core, including the advanced functionality that distinguishes c-tree from other

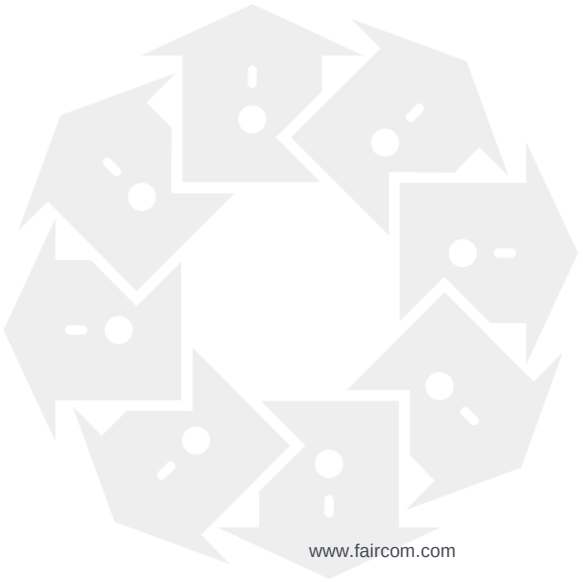
database solutions. FairCom DB API Java offers a method of database creation and maintenance easier to use than traditional c-treeACE ISAM and c tree LowLevel APIs.

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

Using the same relationships as the FairCom DB API interface technology, FairCom DB API Java consists of a Java API that provides the classes and methods that comprise complete database functionality.

10. New Deferred File and Index Maintenance

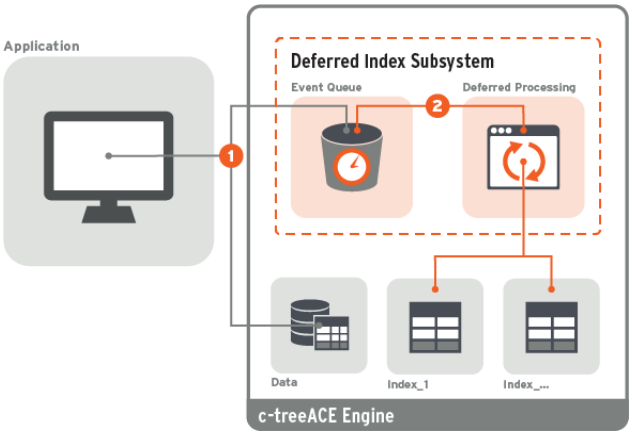
Deferred file and index maintenance postpones index and data maintenance operations, thus offloading performance bottlenecks.



10.1 Deferred Indexing

Deferred Index Maintenance brings new speed to index maintenance operations.

c-tree's ISAM add, delete, and update operations dutifully perform key insert and deletes on every index associated with a data file: a record add inserts a key value for each associated index; a record delete deletes a key value for each associated index; and a record update deletes the old key and inserts the new key for each associated index (for those keys whose values changed). However, each additional index operation can impose measurable performance loss, especially when numerous indexes are involved. If these immediate key operations could be held off for a period of time, applications can gain substantial performance in many circumstances.



Two application usages are considered strong candidates for deferred index processing.

- Deferred build of a permanent index added to an existing table. This typically requires waiting for an index to be loaded with all key values before returning control to the application. If this process could be sent to a background task, and only once the index build was complete does it become available. This index will be acting as "eventually consistent" until the end of the creation and initial population with existing data and data added during this initial period of time. After all keys are loaded it behaves either in synchronous or deferred mode depending on its permanent assigned deferred attributes.
- A particularly interesting application usage is an application deployed with a core set of primary indexes and allows additional end user alternate indexes to be created. With so many indexes, core application performance can be adversely impacted. If the additional index maintenance overhead could be deferred, core application performance remains less impacted while using its primary indexes.

FairCom addressed these challenges with new deferred maintenance modes. By delaying select index operations, applications can quickly create and/or update files with large numbers of indexes very quickly.

A key background support for this functionality is the ability to create an index file with the data file open in shared mode.

Permanent Deferred Index Mode

First addressing the second described scenario. A deferred indexing attribute can be specified on new index creation and enables ISAM record add, delete, and update operations to delay key insert/delete operations for that index file. OR in the *KTYP_DFRIDX* bit into the `ikeytyp` field of that index's *IIDX* structure that you pass to the **CREIFIL()** function.

ISAM operations now avoid additional overhead of directly updating these deferred indexes. With deferred indexing enabled, a background thread performs key insert and delete operations on the deferred index files asynchronously. This mode becomes a permanent attribute of the index. That is, it always remains in this deferred mode.

Background Load of a Regular or Deferred Index

A permanent (as compared to a temporary) index can be created as a regular or permanent deferred index with the data file open in shared mode by setting the *dxtdsiz* field of the *IFIL* parameter passed to **PRMIIDX()** to one of the following modes:

1. *ctNO_IDX_BUILD*

   If using *ctNO_IDX_BUILD*, the index can be loaded later by either:

   a. Calling **RBLIIDX()** with the data file open in exclusive mode (which is already supported), or

   b. Calling **ctDeferredIndexControl()** (page 176) with opcode of *DFKCTLqueueload* to queue the index load to the background index load thread.

2. *ctQUEUE_IDX_BUILD*

   With *ctQUEUE_IDX_BUILD* defined, the index load is immediately queued to a background index load thread after the index is created.

Note that when a *regular* index (non-deferred) is created with the data file open in shared mode, all connections that already have the associated data file open will internally open the new index and their ISAM record add/delete/update operations will affect this new index. This means that if an error occurs when updating the new index (for example if the add or update attempts to add a key that already exists in that index), the operation fails with that error. By contrast, updates to a deferred index that is created with the data file open in shared mode are queued to a background thread.

Background Thread Processing

For files under full transaction control (*ctTRNLOG*), deferred operations are written to transaction logs. A background thread reads the operations directly from the transaction log, calling and optional callback function, and optionally applies the operations to any deferred indices.

For atomicity only files (*ctPREIMG*) and non-transaction-controlled files, deferred operations are written to an in-memory queue. A background thread reads operations from the queue, calling an optional callback function, and optionally applies the operations to any deferred indices.

Note: Deferred Index features are not available for older compilers: Visual Studio VS2005 and older.

Transaction Log Limit for Deferred Indexing

Note: This is a Compatibility Change for c-treeACE V11.5 and later.

The deferred index processing thread for *TRNLOG* files reads transaction logs and informs the server of their minimum transaction log requirements.

c-treeACE Server now limits by default the number of active transaction logs that are kept for deferred index processing to 50. This default was chosen to avoid potentially running out of critical drive space.

The following configuration option was introduced for controlling this limit:

- `MAX_DFRIDX_LOGS <max_logs>` - Maximum number of logs to be held specifically for the deferred indexing thread.

This configuration setting does not impact your c-treeACE Server's ability to retain any necessary logs required for Automatic Recovery.

For more information about this setting (and the related setting for Replication Agent logs), see the following in the *c-treeACE Server Administrators Guide*:

- Transaction log limit for replication and deferred indexing (https://docs.faircom.com/doc/ctserver/69967.htm)
- MAX_DFRIDX_LOGS (https://docs.faircom.com/doc/ctserver/max-dfridx-logs-config.htm)
- MAX_REPL_LOGS (https://docs.faircom.com/doc/ctserver/max-repl-logs-config.htm)

Monitoring and Controlling Deferred Indexing

A Deferred Indexing API (page 176) for monitoring and controlling deferred indexing is available.

A complete Deferred Indexing utility, **dfkctl** (page 227), is also available with many options ready to use.

For more information, see the **ctDeferredIndexControl()** (page 176) function.

Queuing an Index Load

This example shows how to use **ctDeferredIndexControl()** to queue an index load to a background index load thread. The data file must be open by the calling connection. Set the *datno* field of the *DFKQIL* structure to the data file number. The *idxlst* field of the *DFKQIL* structure holds the file numbers of the indexes for which you want to queue an index load operation.

```
DFKQIL  dfkqil;
LONG    keynos[1];

memset(&dfkctl,0,sizeof(dfkctl));
memset(&dfkqil,0,sizeof(dfkqil));

dfkctl.verson = DFKCTL_VERS_V01;
dfkctl.opcode = DFKCTLqueueload;
dfkctl.verson = DFKCTL_VERS_V01;
dfkctl.opcode = DFKCTLqueueload;
dfkctl.bufsiz = sizeof(dfkqil);
dfkctl.bufptr = (pTEXT) &dfkqil;

dfkqil.verson = DFKQIL_VERS_V01;
dfkqil.datno = datno;
dfkqil.numidx = 1;
keynos[0] = datno + 1;
dfkqil.idxlst = &keynos;

if ((rc = ctDeferredIndexControl(&dfkctl)) != NO_ERROR) {
        printf("Error: Failed to schedule index load: %d\n",
                rc);
        goto err_ret;
}
printf("Successfully scheduled index load.\n");
```

Counting the Number of Deferred Operations

A count of the number of operations that have been deferred for the index is stored in the index file's header. A new extended **GETFILX()** API function has been introduced with additional parameter options and can be used to read this value with the *DFRKOPS* mode.

Function prototype

```
ctCONV NINT ctDECL GETFILX( FILNO filno,COUNT mode, pVOID bufptr, pNINT pbuflen );
```

Example

```
LONG8   ndfrkops;
NINT    rc, buflen = sizeof(LONG8);

if ((rc = GETFILX( keyno, DFRKOPS, &ndfrkops, &buflen))) {
    printf("Error: Failed to get deferred operation count: %d\n",
    rc);
} else {
    printf("Number of deferred index operations :" ctLLnd(10) " ",
      ndfrkops);
}
```

10.2 Asynchronous Record Update Notifications

Many c-tree applications store documents and other non-trivial data types. c-tree allows nearly any type of data to be stored, including text, XML, JSON, PDFs, word processing documents, and email. Frequently, these types of data require alternate indexing algorithms compared to how standard data types are indexed with basic b-tree algorithms. What is needed is a method to hand off these types of record updates to alternative handling.

In V11 and later, a set of Record Update Notification callback functions allows applications to externally process records as they are added or updated within a c-tree database.

These callback functions are called when records in the file are modified (added, updated, and deleted) and are asynchronously processed and persisted. Contrast this with the c-treeACE notification feature which is a memory-based queue. While standard notification can trigger callback handling, this is done in a synchronous manner which can impact up-front application performance. The record update notification callback also handles change events, however, its intent is to allow user-defined actions to occur when a record in a particular table is modified, in a deferred asynchronous manner. With this deferred handling, update events are deferred for background processing while maintaining application responsiveness.

Implementing Record Update Notifications

Function prototypes to handle record updates are located in the following c-tree source module:

ctree\source\ctrucbdll.c

To create a functional shared library, build the module using the c-treeACE build utility **mtmake** and choose the c-treeACE *CTUSER* model option. *ctuser.dll* (or *libctuser.so*) is generated and this module is then copied into your server's binary folder location.

The callback feature defines three functions:

1. **File open** callback function called when a connection opens a file,
2. **File close** callback called when a connection closes or deletes a file, and
3. **Record update** callback called when an ISAM-level record add, update, or delete operation is performed on a file.

The **ctRecordUpdateCallbackControl()** (page 171) function, described below, is used to add and delete callback function definitions. A file can have more than one callback function definition. Each callback function definition is identified by its name, which is a case-sensitive ASCII string.

Each callback definition consists of the following attributes:

- Callback physical .DLL name
- Callback logical identifier name
- Callback function names for file open, file close, and record update
- Parameter string (optional)
- Callback time: this is the time when the record update callback is called. Supported options are:
    a. called during a record update
    b. called when a transaction commits (available only for a transaction-controlled file; for a non-transaction-controlled file, this option causes the callback to be called during a record update)

c. written to transaction log or a memory queue and processed by a background thread after a transaction commits or the operation completes
d. written to transaction log or a memory queue and no further action taken: the application is responsible for processing any queued entries

Restrictions

1. c-treeACE allows a callback function definition to be added when the data file is open in shared mode. In standalone mode, adding a callback function requires the data file to be open in exclusive mode.
2. A callback function definition can only be deleted when the data file is open in exclusive mode.

Management API Function

- **ctRecordUpdateCallbackControl()** - Callback definition management API (page 171)

See also:

- **Update Callback Specifications** (page 102)
- Option to specify external library name in platform-independent format

Update Callback Specifications

File Open and File Close Callbacks

The file open and close callback functions are intended to allow an application to manage resources that will be used by the record update callback function. Typically an application will allocate resources in the file open function and will free the resources in the file close function. For example, initializing an alternative indexing environment on open, and freeing that environment context at close.

The file open callback function is called when the following events occur:

1. When a connection opens a file that has a callback definition, the callback function is called at the end of the file open operation. If it is an ISAM level file open call such as **OPNIFIL()** or **OPNRFIL()**, the callback is called after the data file and its associated indexes have been successfully opened.
2. When a connection adds a callback definition to the file, the callback function is called after the callback definition resource in the data file has been successfully updated.

A callback that is added to a data file that is open in shared mode becomes visible to other connections that already have the file open on their next record add, update, or delete operation. In that situation, the other connection finds that the new callback exists and it calls the file open callback function for the newly-added callback before calling the record update callback function.

The file close callback function is called when the following events occur:

1. When a connection closes or deletes the file, the callback function is called right before the data file is closed, so the connection still has the data file and its associated index files open.
2. When a connection deletes a callback definition from the file, the callback function is called after the callback definition resource in the data file has been successfully updated.

The file open and close functions have the following prototypes:

```
NINT rucbOpenFileCallback(pRUCBF prucbf);
NINT rucbCloseFileCallback(pRUCBF prucbf);
```

The record update callback function parameter structure, *RUCBF*, has the following definition:

```
/* record update callback parameters */
typedef struct rucbf {
    pTEXT   datnam;    /* Data file name.                          */
    pTEXT   params;    /* Optional callback parameter string.      */
    NINT    calltm;    /* Current context for this call.           */
    FILNO   datno;     /* User file number of data file.           */
    pVOID   psession;  /* User-defined connection-level pointer.   */
    pVOID   ptable;    /* User-defined table-level pointer.        */
} RUCBF, *pRUCBF;
```

The two state pointers, *psession* and *ptable*, are available for use by the user-defined callback function code.

The *psession* state pointer is shared by all callback functions for all files in a given connection. It is appropriate for storing connection-wide state information. One way to use this pointer is to allocate memory and set *psession* to point to that memory on the first call to the file open callback function in a connection. The application can also maintain a connection-wide reference count, and when the file close function finds that the reference count is zero, it can free the memory.

The *ptable* state pointer is specific to a particular callback function for a particular file. It is appropriate for storing table-level state information.

Record Update Callback

When a user adds, updates, or deletes a record at the ISAM level, the callback is called at one of the following events:

1. If the callback event is set to *RUCBonrecupd*, or if the callback time is set to *RUCBontrancmt* and the file is not under transaction control, the callback is called right after the record and its keys have been added, updated, or deleted.
2. If the callback event is set to *RUCBontrancmt* and the file is under transaction control, the callback is called when the transaction that modified the record is committing.
3. if the callback event is set to *RUCBonqueuethr*d, the callback is called after the transaction commits, when the deferred index thread has read the entry for that record modification operation from the transaction logs.
4. if the callback event is set to *RUCBonqueueap*p, the callback is never called. It is up to the application to read the entry for that record modification operation from the transaction logs and take the appropriate action.

The record update callback function has the following prototype:

```
NINT rucbRecordUpdateCallback(pRUCBF prucbf,pDFRKY pdfrky);
```

The *RUCBF* structure is the same as mentioned above and the additional *DFRKY* structure contains the information about the record add, update, or delete operation and has the following definition:

```
typedef struct dfrky {
    COUNT   opcode;     /* deferred index operation code           */
    TEXT    status1;    /* status bit field #1                     */
    TEXT    status2;    /* status bit field #2                     */
    ULONG   dfrkctr;    /* deferred index create counter           */
    LONG    fid[3];     /* unique file ID of data file             */
    LONG    oreclen;    /* size of old record image (rewrite)      */
    LONG    reclen;     /* size of record image                    */
    LONG    datnamlen;  /* length of data file name plus null      */
    LONG8   orecbyt;    /* old record offset (rewrite)             */
    LONG8   recbyt;     /* record offset                           */
    TEXT    varinf[1];  /* variable length information:
                        ** For ctDFR_ADDKEY and ctDFR_DELKEY:
                        **   null-terminated data file name
                        **   old record image (rewrite)
                        **   record image
                        ** For ctDFR_RWTKEY and ctDFR_RWTPKEY:
                        **   null-terminated data file name
                        **   old record image
                        **   record image
                        ** For ctDFR_LOADKEY:
                        **   null-terminated data file name
                        **   null-terminated index file name
                        */
} DFRKY, ctMEM **ppDFRKY;
```

Your record update callback implementation handles one of following four update operations provided in the opcode from the *DFRKY* structure:

- *ctDFR_ADDKEY* - A record was added
- *ctDFR_RWTKEY* - A record was updated
- *ctDFR_RWTPKEY* - A record was partially updated
- *ctDFR_DELKEY* - A record was deleted

You'll find this framework available in the *ctrucbdll.c* module.

First and Last Operations

An application that uses the record update callback function feature needs to know what operations are associated with a particular transaction and what are the first and last operations for that transaction. In V11.5 and later FairCom Server provides this information to the record update callback function:

- The first operation in a transaction has the ctDFR_TRANFRS bit set in the status1 field of the *pdfrky* parameter that is passed to the record update callback function.

  The last operation in a transaction has the ctDFR_TRANLST bit set in the status1 field of the *pdfrky* parameter that is passed to the record update callback function. So, if a transaction has only one DFRKEY entry, that entry will have both the ctDFR_TRANFRS bit and the ctDFR_TRANLST bit set. If a transaction has more than one DFRKEY entry, the first entry will have the ctDFR_TRANFRS bit set and the last entry will have the ctDFR_TRANLST bit set.

- An optional parameter of type pRUCBSTT is passed to the record update callback function. This structure has the following definition:

```
typedef struct rucbstt_t {
    LONG    verson; /* structure version   */
    LONG    avail;  /* padding- available for use  */
    LONG8   tranno; /* transaction number for the operation */
} RUCBSTT, *pRUCBSTT;
```

Note: For FairCom Server to pass this third parameter to your record update callback function, your record update callback DLL or shared library must export the function **rucbCheckVersionCallback()**, which has the following function prototype:

```
NINT rucbCheckVersionCallback(pRUCBACB prucbacb,pNINT pversion);
```

prucbacb is the record update callback definition in the format of the record update callback add operation structure, RUCBACB. The function can choose to examine the record update callback definition to decide which version of the RUCB API it supports. For example, it can choose based on the callback name (`prucbacb->cbname`) or the name of the record update callback function (`prucbacb->fncnames[2]`).

The function should set *pversion* to the version of the record update callback API your DLL uses and return zero to indicate success. Supported versions are:

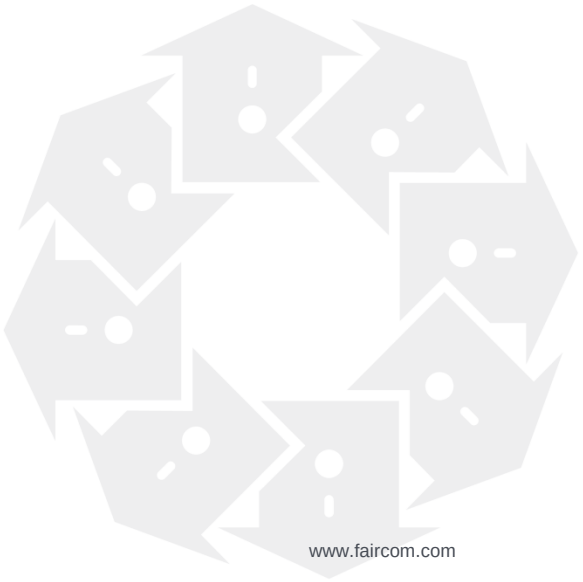
- a) Version 1 of record update callback API. Your record update callback function must conform to the following prototype:
  `NINT rucbRecordUpdateCallback (pRUCBF prucbf,pRUCBO prucbo);`
- b) Version 2 of record update callback API. Your record update callback function must conform to the following prototype:
  `NINT rucbRecordUpdateCallback (pRUCBF prucbf,pRUCBO prucbo,pRUCBSTT prucbstt);`

If your record update callback DLL does not export a function named **rucbCheckVersionCallback()**, FairCom Server uses version 1 of the record callback API.

11. Coordinate Application Recovery with Transaction Restore Points

Now c-treeACE gives you greater control in balancing data integrity against performance and recovery.

A Restore Point marks a position in a transaction log to which FairCom Server's automatic recovery can roll back the system in event of system failure.



11.1 Transaction Restore Points

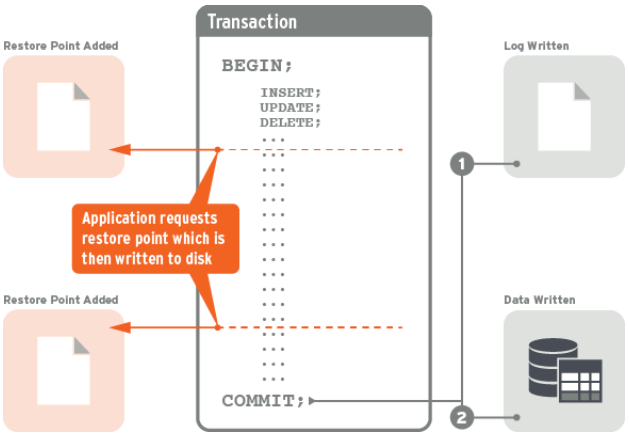
Transaction control by definition requires logging all database changes to persisted storage. Of course, transaction control is highly desirable for recovery of the database in the event of system failure. It also makes available other valuable features such as replication. However, this comes with a direct and measurable performance cost.

The new Delayed Durability feature can greatly enhance performance; however, should a system fail, database recovery could take considerable time if the transaction logs held substantial volumes of data not yet flushed to data and index files.

This leads to an additional challenge of how to balance data integrity using transactions against performance **and** recovery. Certain classes of applications can restore to known points in time. Consider the simple case of a bulk load. Should anything fail during loading, the load process can usually be restarted. If this known position could be coordinated with a c-treeACE server transaction log position, one could architect a very effective recovery process.

To this end, a new Restore Point feature has been introduced. A Restore Point is a position in a transaction log to which FairCom Server's automatic recovery can roll back the system in event of system failure.

The Restore Point is used for restoring a system to a point in time coordinated with the application as a known good state. After restoring the system to a Restore Point, the application can be architected to perform work that was not recovered. For example, those applications maintaining their own journal of operations. By coordinating a known good state in time between the application and the server's transaction logs, these applications can quickly recover back to a point in time when absolutely needed.



See **Transaction Restore Points for Application Recovery**.

Lightweight vs. Checkpoint Restore Points

By reducing the need to sync the transaction log to disk, this new transaction processing mode can improve performance substantially. The tradeoff is that automatic recovery is not guaranteed to make good on all the committed transactions. To make recovery predictable and usable, we introduce the concept of a Restore Point (RP). A Restore Point is a place in a transaction log that has no active transactions. Two types of Restore Points are available:

- **Lightweight Restore Point:** A **RSTPNT** (restore point) log entry is a placeholder where no transactions are active.
- **Checkpoint Restore Point:** A **CHKPNT** (check point) log entry is a place at which no transactions are active.

Every *N* minutes (where *N* is a small number), the application will issue a lightweight checkpoint, and before restarting transactions, start an external activity log. If a crash occurs, it is possible to recover to the last Restore Point, and redo the activities in the external log.

A Checkpoint Restore Point provides a clean place in the log where roll forwards from a Dynamic Dump can stop and restart. Imagine that a system in operation runs a complete dynamic dump, and then periodically (each hour, each day, etc.) issues a Checkpoint Restore Point. A backup system can be maintained in fairly up-to-date fashion by using the Checkpoint Restore Points as convenient milestones to roll forward from and roll forward to.

11.2 Creating Restore Points

Restore Points are created using the **ctQUIET()** API function. A Restore Point can be either a Lightweight Restore Point (*RSTPNT*) or a Restore Point with a full Checkpoint Restore Point (*CHKPNT*).

Lightweight Restore Point

To establish a Lightweight Restore Point, call:

```
ctQUIET(NULL,timeout,ctQTlog_restorepoint)
```

This call blocks new transactions from starting and waits for up to *timeout* seconds before aborting pending transactions.

The *timeout* parameter indicates the number of seconds **ctQUIET()** will wait for transactions to complete before aborting active transactions. If all the active transactions complete before *timeout* seconds, then no transactions will be aborted and the Restore Point will be written as soon as possible. When **ctQUIET()** is called, new transactions are immediately blocked, and **ctQUIET()** will not write the Restore Point in the log until all transactions complete or are aborted.

When all transactions have been committed or aborted, it writes a log entry with transaction type of *RSTPNT*. The log entry contains the time of the Restore Point and the name of the file that contains the log number and offset of the Restore Point entry. The formatting and naming of the Restore Point file is explained in *Restore Point Files* (page 119).

FairCom Server guarantees that each Restore Point has its own unique timestamp by ensuring that successive Restore Points have a minimum interval of at least 5 seconds. As a matter of

efficiency, we would expect the actual interval between Restore Points to be much longer than a few seconds; more likely an interval on the order of 5 minutes or longer. The interval is under the control of the application.

The advantage of creating a Restore Point using *ctQTlog_restorepoint* without *ctQTlog_checkpoint* is that there is much less overhead writing a short *RSTPNT* log entry compared to creating and writing an entire checkpoint.

Checkpoint Restore Point

To create a Restore Point with a full checkpoint, call **ctQUIET()** with both *ctQTlog_restorepoint* and *ctQTlog_checkpoint*:

```
ctQUIET(NULL,timeout,ctQTlog_restorepoint | ctQTlog_checkpoint)
```

As with a lightweight Restore Point, this call blocks new transactions from starting and waits for up to *timeout* seconds for existing transactions to complete, after which it aborts all transactions that are still pending. Then it writes a checkpoint (with a mode of RESTORE_CHECK) to the transaction log instead of writing a *RSTPNT* entry and creates a Checkpoint file. The formatting and naming of the Checkpoint file is explained in *Restore Point Files* (page 119).

Note: *ctQTlog_restorepoint* and *ctQTlog_checkpoint* cannot be used with any other blocking modes.

Permitting New Transactions

After the **ctQUIET()** call returns successfully (after either a Lightweight Restore Point or a full Checkpoint Restore Point), call **ctQUIET()** with an action of either *ctQTunblockLogRestore* or *ctQTunblockALL* to permit new transactions:

```
ctQUIET(NULL,timeout,ctQTunblockLogRestore)
```

Before calling **ctQUIET()** a second time to unblock new transactions, it may be desirable to start an external (i.e., separate from FairCom Server) activity log managed by the application. This log can be used to recover updates made after the Restore Point.

Notes

A **ctQUIET()** call to establish a Restore Point cannot be mixed with any other blocking actions. An application that attempts a TRANBEG after **ctQUIET()** has been called to establish a Restore Point will be blocked: the TRANBEG will not return from the server until the second call to **ctQUIET()** is made. An application whose transaction is aborted by the first **ctQUIET()** call will receive an error return of **QTAB_ERR**.

Note: A minimum interval of 1 second is enforced between calls to establish a Restore Point. While it is very unlikely that anyone would make calls to establish Restore Points with a frequency even approaching 1 second, the **ctQUIET()** logic ensures the interval will be at least 1 second. A call to **ctdefer()** sleeps the **ctQUIET** thread if necessary to enforce the time interval.

Note: Unless `COMPATIBILITY NONADMIN_QUIET` is in the configuration file, a Restore Point can only be established by a client in the ADMIN group.

Note: If a client successfully calls **ctQUIET()** to establish a Restore Point and then terminates without unblocking new TRANBEG calls through a second call to **ctQUIET()**, the server will force a second call. Ordinarily the **ctQUIET** blocks (for other types of block requests) persist when the blocking thread terminates without issuing an unblock request.

Active Restore Points

The configuration option `KEEP_RESTORE_POINTS N`, which defaults to 1, specifies the number of Restore Points that FairCom Server considers to be active Restore Points for the purpose of determining which transaction log files must remain active in order to be able to roll back to the *N*th most recent Restore Point. When a new Restore Point is created, the oldest Restore Point is no longer considered active.

Note: To be able to roll back to a particular Restore Point, c-tree must keep the transaction log that contains the checkpoint preceding that Restore Point, plus all subsequent transaction logs. Therefore, if you create a Restore Point and never create another Restore Point, all transaction logs from that point forward will be kept as active logs. So when using Restore Points, it is advisable to *periodically establish new Restore Points so that FairCom Server can delete older transaction log files*.

11.3 Rolling Back to a Restore Point

If FairCom Server terminates abnormally when there are active transactions or there are open updated transaction-controlled files and it is restarted with the option `RECOVER_TO_RESTORE_POINT YES` in *ctsrvr.cfg* then, after FairCom Server completes its automatic recovery, it rolls the transaction-controlled data and index files back to the most recent Restore Point.

The following messages in *CTSTATUS.FCS* indicate that FairCom Server successfully rolled back to a Restore Point:

```
- User# 00001      Rollback to Restore Point requested
- User# 00001      Recovery is rolling back to Restore Point ...
- User# 00001      20140707_155104
- User# 00001      Scanning transaction logs
- User# 00001      Recovery rolled back to Restore Point ...
- User# 00001      20140707_155104
- User# 00001      Automatic recovery completed
```

If a request to roll back to the Restore Point fails because no active Restore Point exists, FairCom Server terminates with error **NORP_ERR** (1015). If an active Restore Points exists, but the transaction log scanning or the transaction undo's fail, the rollback to the Restore Point returns the error, and the server terminates. An example would be the inability to undo a file delete for a *ctRSTDEL* file.

Monitoring Restore Point Activity

When FairCom Server successfully rolls back to a Restore Point, it writes an entry to its system log file. After starting FairCom Server, an administrative application can read the system log to determine if the server has rolled back to a Restore Point.

The *ctalog.c* example program demonstrates how to read records from the system log.

SYSLOG Logging of Restore Point

A new *SYSLOG* class has been added: `ctSYSLOGrstpnt`. The associated configuration file entry is `SYSLOG RESTORE_POINT`.

There are four events associated with the Restore Point class:

- *ctSYSLOGrstpntCREATE* - Create a Restore Point.
- *ctSYSLOGrstpntNOKEEP* - Create a Restore Point, but KEEP_RESTORE_POINTS is 0.
- *ctSYSLOGrstpntRECOVERY* - Automatic recovery with Restore Points.
- *ctSYSLOGrstpntTRANBAK* - Transaction rollback (via utility program) with Restore Points.

SYSLOG Restore Point Record Format

The *SYSLOG* record associated with a Restore Point event starts with the common *SYSLOG* record format and is followed by a variable region. The variable region contains the *SYSLOG* Restore Point information. It is made up of the `syslogRP` structure defined in ***ctport.h***:

```
#define ctRPbvr         24      /* base variable region                */
typedef struct rstpntsyslog {
        ULONG   bitmap;         /* syslog rstpnt bitmap                */
        LONG    varlen;         /* len of var region (may be > ctRPbvr) */
        LONG8   rstpntsrn;      /* Restore Point serial number         */
        LONG    rstpntlog;      /* Restore Point log#                  */
        ULONG   rstpntpos;      /* Restore Point log position          */
        ULONG   rstpnttim;      /* Restore Point system time           */
        LONG    rstpnttim2;     /* system time extension               */
        LONG    skplog;         /* skip forward log number             */
        ULONG   skppos;         /* skip forward log position           */
        ULONG   avail;          /* available                           */
        ULONG   temptim;        /* time event added to temp event file  */
        LONG    temptim2;       /* temptim extension                   */
        UCOUNT  rstpntver;      /* Restore Point version               */
        UCOUNT  rstpnttyp;      /* Restore Point type                  */
        TEXT    var_region[ctRPbvr]; /* beginning of var region        */
        } syslogRP;
```

The `var_region` in the `syslogRP` structure is the base region for any additional variable-length data. The `varlen` member of `syslogRP` indicates how much variable data is in the record and may exceed `ctRPbvr` as defined above.

The most important component of this record for a `ctSYSLOGrstpntRECOVERY` event is the bitmap member that indicates the details of what happened during a recovery that involves Restore Points and/or rolling back to a Restore Point. *ctport.h* has the bitmap values. They are:

```
#define syslogRPname            0x000001  /* RP name in var region                  */
#define syslogRPminlogsync      0x000002  /* Deferred LOG SYNC feature on at crash  */
#define syslogRPrec_to_rstpnt   0x000004  /* RECOVER_TO_RESTORE_POINT on            */
#define syslogRPstart_rolbak    0x000008  /* rollback attempted                     */
#define syslogRPpndg_rolbak     0x000010  /* rollback to Restore Point pending      */
#define syslogRPno_rstpnt       0x000020  /* NO Restore Points                      */
#define syslogRPno_rolbak_sup   0x000040  /* rollback not supported                 */
#define syslogRPno_tran_undo    0x000080  /* may be trans not undone                */
#define syslogRProlbak_err      0x000100  /* rollback error                         */
#define syslogRPskipto_err      0x000200  /* error inserting skipto info            */
#define syslogRPdef_not_rstpnt  0x000400  /* RECOVER_TO_... NO by default           */
#define syslogRPskip_pndg       0x000800  /* skip pending rollback                  */
#define syslogRPno_tran_to_undo 0x001000  /* no trans to undo, skip rolbak          */
#define syslogRProlbak_rstpnt   0x002000  /* rollback to Restore Point              */
#define syslogRPno_fnd_rstpnt   0x004000  /* did not find Restore Point             */
#define syslogRPskip_pndg_sync  0x008000  /* skip pending rollback && MLS           */
#define syslogRPpndg_rolbak_sync 0x10000  /* rollback to Restore Point pending && MLS*/
#define syslogRProlbak_not_rqst 0x020000  /* rollback to Restore Point not requested */
#define syslogRPavailable       0x040000  /* available for use                      */

 /*
 ** final system state indicators (MLS stands for Deferred/Minimum LOG SYNC feature)
 */
#define syslogRP_FSrolbak       0x0080000  /* successful rollback to Restore Point  */
#define syslogRP_FSno_rolbak_OK 0x0100000  /* no rollback to Restore Point. no MLS. */
#define syslogRP_FSrolbak_err   0x0200000  /* rollback error                        */
#define syslogRP_FSno_rolbak_MLS 0x400000  /* no rollback to Restore Point, but MLS */
#define syslogRP_FSrolbak_NRP   0x0800000  /* rollback did not find Restore Point   */
#define syslogRP_FSrolbak_chg   0x1000000  /* rollback NORP. try explicit **
                                           ** RECOVER_TO_RESTORE_POINT NO */
#define syslogRP_FSrolbak_chg2  0x2000000  /* rollback NORB. try explicit **
                                           ** RECOVER_TO_RESTORE_POINT NO **
                                           ** or YES                      */
#define syslogRP_FSrolbak_err2  0x4000000  /* rollback error on skipto upd          */
```

The final system state indicators, of which only one should be defined for each recovery event, indicate the state of the server after recovery. In the case of an error, the error code will be in the common *SYSLOG* record header. The **ctalog** utility section (page 113) provides more expansive descriptions and the significance of each bitmap value. The final system state indicators provide an overview of the "health" of the server and its data files.

Temporary Event File

System events are stored in *SYSLOGDT.FCS* and *SYSLOGIX.FCS* which are c-tree ISAM files. When a server is starting and automatic recovery occurs involving Restore Points, a `ctSYSLOGrstpntRECOVERY` event is logged. However, if a startup error occurs, the server

comes down before the *SYSLOG* files are available. In this case, the events are stored in a temporary event file named *EVENT_DT.FCS*. This file is almost identical to *SYSLOGDT.FCS*. The difference is that each record starts with the log position of the last checkpoint. This information is used in subsequent processing of the temporary event file. This file is created if needed, and typically does not exist.

Once a server makes a successful start, it checks for the temporary event file, reads each record in physical order, and if the starting checkpoint for recovery agrees with the checkpoint location stored in the *EVENT_DT.FCS* record, the event is added to the *SYSLOG* files. At the end of reading all records, the event file is renamed to the following (where the date and time correspond to the system time when the temporary file is renamed):

EVENT_DT.FCS.YYYYMMDD_HHMMSS

The reason more than one record may exist in the temporary event file is that successive failed startups will each contribute a record until the server startup is successful, at which time it is renamed. The renamed files are not used by the server, but they are available for archival purposes.

Using ctalog SYSLOG Utility to Read Restore Point Data

The **ctalog** utility is a sample program that can read the *SYSLOG* files and display their contents. It can purge the contents of the *SYSLOG* files. A sample record output from **ctalog** is shown below.

The first record corresponds to a failed server startup because of a read error (**36**) with the temporary event file. (Such an error is very unlikely and was forced under a debugger for demonstration purposes.) The second record output shows the result of the subsequent successful startup with a rollback. Note that the second record output indicates a rollback is pending. The rollback is pending because the first attempt at recovery failed.

```
Class           = 8 (Restore Point)
Event           = 2 (Recovery results with Restore Points)
Date            = 08/08/2014
Time            = 11:57:19
Sequence number = 2
Error code      = 36
User ID         = ''
Node name       = ''
Variable-length information:
--------------------------------------------------

        Restore point specifications -
        serial number:          1
           log number:          1
         log position:      23e58x
            timestamp: 1407499816  Fri Aug 08 07:10:16 2014
                 type: Lightweight RP
                 name: 20140808_071016

        Recovery/Restore Point status bits:
          - Deferred LOG SYNC was enabled at the time of the crash
          - RECOVER_TO_RESTORE_POINT is enabled
          - rollback attempted
          - rollback error
          - error inserting skip forward info into log

        Event originally added to temporary event file on
        failed startup at 20140808 115705

        System state:
          All transactions committed before the system crash are recovered, but
rollback to a Restore Point failed (see error code) during the insertion of the
skip forward information into the Restore Point. It may be necessary to use the
log file copied before the insertion.

--------------------------------------------------

Class           = 8 (Restore Point)
Event           = 2 (Recovery results with Restore Points)
Date            = 08/08/2014
Time            = 11:57:19
Sequence number = 3
Error code      = 0
User ID         = ''
Node name       = ''
Variable-length information:
--------------------------------------------------

        Restore point specifications -
        serial number:          1
           log number:          1
```

```
         log position:      23e58x
            timestamp: 1407499816  Fri Aug 08 07:10:16 2014
                 type: Lightweight RP
                 name: 20140808_071016

        Recovery/Restore Point status bits:
          - Deferred LOG SYNC was enabled at the time of the crash
          - pending rollback from previous recovery from Deferred LOG SYNC crash
          - RECOVER_TO_RESTORE_POINT is enabled
          - rollback attempted
          - rollback to Restore Point completed

        System state:
          All transactions committed before the Restore Point are recovered, and
 any transactions committed after the Restore Point are undone.

-------------------------------------------------
```

The system state output corresponds to the final system state bitmap entries shown above. The following routine is in *ctalog.c*:

```
VOID printRPsyslogrec (pSYSLOGrec psr,NINT print_header)
```

This routine could be used in other applications designed to read the *SYSLOG* files. Its only external requirements are the **dateout()** and **timeout()** routines included in *ctalog.c*.

The *print_header* parameter indicates whether or not to print the information from the common *SYSLOG* record header. **ctalog** sets this to NO since it already prints the information for each event. For a specialized *SYSLOG* reader, one might want to print the header information from the routine. **ctalog** calls this routine for each `ctSYSLOGrstpnt` event.

11.4 Automatic Recovery Considerations

As noted previously, Delayed Durability affects automatic recovery. The following outcomes of automatic recovery are possible after a server crash that occurred when the feature was enabled:

- *Active Restore Points exist and* `RECOVER_TO_RESTORE_POINT` *is* `YES`:
  All transactions committed before the Restore Point are recovered, and any transactions committed after the Restore Point are undone.
- *Active Restore Points exist and* `RECOVER_TO_RESTORE_POINT` *is* `NO`:
  All transactions committed before the Restore Point are recovered, but some transactions committed after the Restore Point may have been recovered and others lost.
- *Active Restore Points exist and* `RECOVER_TO_RESTORE_POINT` *is not in the configuration file*:
  All transactions committed before the Restore Point are recovered, but some transactions committed after the Restore Point may have been recovered and others lost; then the server terminates with a **NORB_ERR**. The next server startup will detect that a rollback to the last active Restore Point is pending. Add `RECOVER_TO_RESTORE_POINT YES` or `NO` to the configuration to successfully restart the server.

Note: If automatic recovery has completed the first stage of recovery (i.e., all transactions committed before the crash whose log entries made it to disk are recovered), but the server does not complete the second stage of rolling back to the last Active Restore Point; then the next server startup will detect the pending rollback.

Upon successful rollback to the last Restore Point, the Restore Point log entry is modified to include the skip forward location in the log. The skip forward location stored in the Restore Point permits a roll forward operation to skip over transactions that have been undone because of a prior rollback to the Restore Point.

Note: The modification of the Restore Point log entry is the only time c-tree changes an existing log entry. To avoid permanently corrupting the log containing the Restore Point in the unlikely event that the log update operation fails, the log is copied before attempting to modify the Restore Point entry. The copied log has a name in the form:

`LNNNNNNN.FCS.YYYYMMDD_HHMMSS`

where `NNNNNNN` is the log number, and the date and time in the name correspond to the system time at which the log was copied. These copied log files are NOT deleted by the server.

11.5 Rollback to New Restore Points with ctrdmp

In V11 and later, **ctrdmp** is able to rollback to a Restore Point. Restore Points permit server clients to establish quiet spots in the transaction log where there are no active transactions.

Prior to the V11 modifications, **ctrdmp** could either perform a dynamic dump recovery or rollback to a specified date and time. **ctrdmp** has been extended such that, as an alternative to specifying a date and time, the rollback script can provide the name of a Restore Point file.

A typical **ctrdmp** script file used for a rollback looks like:

```
!ROLLBACK
!DATE MM/DD/YYYY
!TIME HH:MM:SS
....
```

Now the script can be composed as follows:

```
!RP <Restore Point File Name>
....
```

The `Restore Point File Name` generated by the server is either of the following:

- *RSTPNT_NO_CHK.YYYYMMDD_HHMMSS.FCS* for a Lightweight Restore Point
- *RSTPNT_CHKPNT.YYYYMMDD_HHMMSS.FCS* for a Checkpoint Restore Point

Note that, as with the `!ROLLBACK` script keyword, the `!RP` keyword must be the first entry in the script file.

See also

- *ctrdmp - Dynamic Dump Recovery or System Rollback* (https://docs.faircom.com/doc/*ctserver/ctrdmp-util.htm*)
- *ctfdmp - Forward Dump Utility* (https://docs.faircom.com/doc/*ctreeplus/ctfdmp-util.htm*)

11.6 Restore Points as an Incremental Roll Forward Strategy

Restore Points add significant value to the Dynamic Dump hot backup feature and long term backup maintenance.

To reduce the overhead of Dynamic Dumps, especially when the Dynamic Dumps are run routinely to provide a backup mechanism, the Restore Point feature allows a single Dynamic Dump to be followed by RPs, eliminating the need for additional Dynamic Dumps.

The RP capability introduced support for two types of RPs:

- Lightweight Restore Point (RP-LW) for use with this option.
- Checkpoint Restore Point (RP-CP) for use with Dynamic Dumps (**ctdump**) and transaction roll forward (**ctfdmp**).

When a Checkpoint Restore Point is created, a Restore Point file is created. This file is very similar to a start file except that it points to the log position of the RP instead of a checkpoint. For more about the file, see *Restore Point Files* (page 119).

Without Restore Points, the **ctfdmp** utility updates c-tree files starting from a checkpoint in the transaction log that has no active transactions, and continues executing until all the subsequent transaction log entries have been processed. Typically, the starting point for the **ctfdmp** is specified by the start file that the Dynamic Dump restore (**ctrdmp**) creates.

Checkpoint Restore Points provide a safe, well-defined position in the transaction log where **ctfdmp** can terminate. It can then be restarted at this same location (using the Restore Point to create a start file), and run until another Checkpoint Restore Point, and so on. Hence an "incremental roll forward" ability.

Don't use superfiles with incremental roll forward. Superfiles are OK with Dynamic Dump, restore, and a single roll forward. Due to how superfiles store offsets, incremental roll forwards with superfiles could cause a problem. Keep in mind that *FAIRCOM.FCS* is a superfile, so we recommend omitting this file if you are performing incremental roll forward operations.

Incremental Roll Forward Strategy

Primary Server System Maintenance

1. Your primary server configuration should include `KEEP_LOGS -1` to ensure all the transaction logs required by your backup maintenance strategy will be available.
2. Run an initial Dynamic Dump to serve as the starting point for your backup maintenance.
3. Periodically create Checkpoint Restore Points, each of which produces a Restore Point file.

Backup Server System Maintenance

1. Copy the Dynamic Dump script file and the Dynamic Dump stream file (containing the dump contents) from above step 2 (Initial Dynamic Dump).

2. Run **ctrdmp** to extract the starting data and index files.
3. Periodically copy completed transaction logs and Restore Point files from your primary server system to the backup server system, and run **ctfdmp** to advance the backup system contents to the latest Restore Point.

In the event your primary server is shut down or crashes and it is desired to switch to the backup server, it will be necessary to run **ctfdmp** from the last restore point until the end of the existing log entries. c-tree files are then available on the backup server ready to be used as your now primary server.

Manually Creating a Point-In-Time Forward Roll

A forward roll can only be started from a checkpoint and not from an arbitrary point in time reached with a rollback operation. Furthermore, the checkpoint must be created when the following conditions are true:

1. no transactions are active;
   and
2. no index buffers contain unflushed updates for committed transactions;
   and
3. no data cache pages contain unflushed updates for committed transactions.

This means that a forward roll requires more than just a starting checkpoint: the state of the data and index files must correspond to the current state of the transaction logs and the position of the starting checkpoint. To roll forward, you will need to have saved a point-in-time copy of the data files, index files, transaction logs, and transaction start files. There are several methods to generate a forward roll eligible backup:

- Using the dynamic dump (**ctdump**), followed by dump restore with the forward roll option (**ctrdmp**)
- A filesystem level backup taken after a clean server shutdown.
- A filesystem level backup taken during a server quiesce with full consistency (**ctquiet** *-f*)

See Also:

- *Restore Points as an Incremental Roll Forward Strategy* in the *V11 Update Guide* (https://docs.faircom.com/doc/*v11ace/64772.htm*)
- *Rolling Forward from Backup* in the *Server Administration Guide* (https://docs.faircom.com/doc/*ctserver/forward-roll.htm*)

11.7 Restore Point Files

Each time a Lightweight Restore Point or Checkpoint Restore Point is successfully established, a dedicated *.FCS* file is written. The Restore Point file contains the location in the log where the Restore Point is written, the system time at which the Restore Point was established, and a sequential serial number assigned to the Restore Point.

Lightweight Restore Point File

Prior to V11.5, the file for a Lightweight Restore Point is named as follows:

RSTPNT_NO_CHK.YYYYMMDD_HHMMSS.FCS

where *YYYYMMDD_HHMMSS* is the date and time at which the Restore Point was created (e.g., *20140804_152014 corresponds to 4 August 2014, 15:20 and 14 seconds*)

The contents of the Restore Point file are formatted similar to a start file (*S0000000/1.FCS*).

File Names in V11.5 and Later

A change has been made in V11.2.3 to simplify determining which transaction logs are required for a restore to use this restore point. The restore point file naming has been updated to include the log number of the checkpoint that the restore point references.

- This affects names of lightweight and checkpoint restore points.

The new formatting is as follows:

```
RSTPNT_CHKPNT.L#######.YYYYMMDD.HHMMSS.FCS
RSTPNT_NO_CHK.L#######.YYYYMMDD.HHMMSS.FCS
```

Where `#######` is the transaction log number of the associated checkpoint.

Note: This is a **Compatibility Change** due to the new naming of restore point files.

Checkpoint Restore Point File

For a Checkpoint Restore Point, prior to V11.5, the file was named *RSTPNT_CHKPNT.YYYYMMDD_HHMMSS.FCS*. (See above for V11.5 and later.)

This Restore Point file is created by **ctQUIET()** by simply copying the start file created by the checkpoint (that does double-duty as a Restore Point). A start file for a Checkpoint Restore Point is the same as an ordinary start file. When rolling forward or backwards, Restore Point files provide a means to pass the target Restore Point information to a utility program.

11.8 Restore Point Limitations

Note the following limitations of these features:

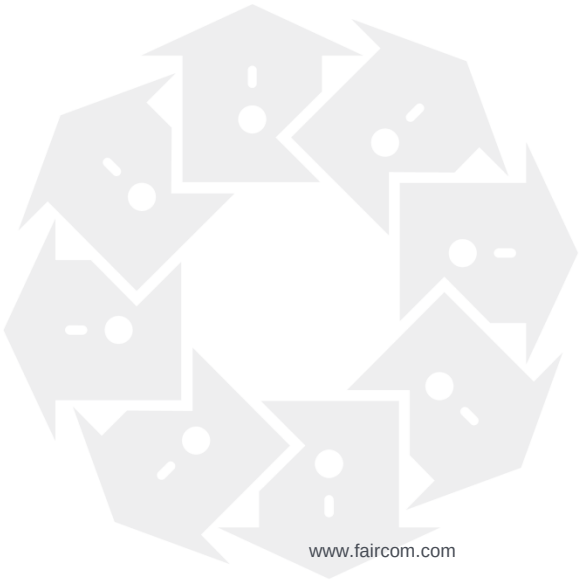
1. FairCom Server remembers the number of currently-active Restore Points based on the `KEEP_RESTORE_POINTS` configuration option and it keeps transaction logs needed to roll back to those Restore Points. However, FairCom Server's automatic recovery only supports rolling back to the most recent Restore Point.
2. File creates, renames, and deletes that are not transaction-dependent (i.e., for files that are not created using the *ctTRANDEP x8mode* attribute) cannot be done when rolling forward or undone when rolling back to a Restore Point. File deletes on files that do not use the

restorable delete (*ctRSTRDEL*) x8mode attribute cannot be undone when rolling back to a Restore Point. The exceptions are files with the *ctTRANDEP x8mode* attribute whose delete is not committed before a system crash. Such files are restorable during the rollback to a Restore Point.

3. The Deferred LOG SYNC feature does not support mirrored transaction log files.
4. c-tree's replication support has no knowledge of Restore Points. If FairCom Server's automatic recovery rolls back to a Restore Point, the Replication Agent will not know if it should skip the transaction log entries that follow the Restore Point (if it has not yet processed them) or undo the transactions that follow the Restore Point (if it has already processed them). To synchronize the source and target data files when the source server rolls back to a Restore Point, the administrator must copy the replicated source data and index files to the target system and then continue replication at the source server's current log position.
5. Incremental roll forward is not supported on superfiles.

12. Embedded Web Server for Browser Based Administration

c-treeACE now includes an embedded web server for remote browser-based administration.



12.1 A c-treeACE SQL Explorer for Your Web Browser

c-treeACE now includes an embedded web server for remote browser administration.

All web components including JavaScript files and CSS templates are contained in a single *\tools* folder found under the working directory of your c-treeACE SQL installation. This is the default location the embedded web service looks for components. This is configurable as the document root configuration.

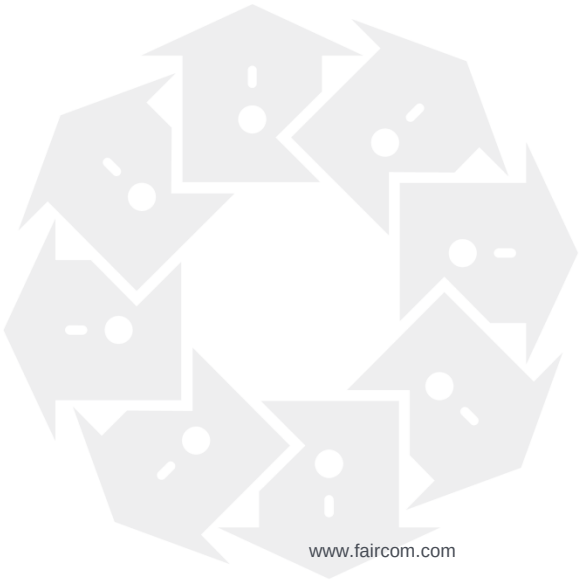
Simply point your browser to *http://<fully.qualified.domain.name>/tools* to access your web service. The embedded web service uses default web port 80. This is also configurable. You'll be presented with a familiar logon prompt.

It is also possible to use your existing Apache or Microsoft IIS web servers. Simply copy the *\tools* folder to your local web server and access from there. Currently, web based c-treeACE server access requires a working c-treeACE SQL PHP installation.

A complete c-treeACE SQL Explorer tool is included at this time. Look for future additions with additional administrative web browser interfaces.

13. Extended c-treeACE Features

This section presents an overview of the enhancements in this release. If you are upgrading from an earlier release, be sure to see ***Notable Compatibility Changes*** *(page 247) for important compatibility information.*



13.1 Millisecond Timestamp Resolution Support

Data acquisition has become much faster and along with it, the necessity for higher time resolution. Due to popular request, we now introduce millisecond timestamp resolution for c-tree time types. This support is available across multiple c-tree APIs, including core c-tree support, FairCom DB API and c-treeACE SQL. This support required addition of extended data types, which you should understand may impact backward compatibility with some applications.

This support includes the following changes:

1. A new data type, *CT_TIME_MS,* was added to the core c-tree DODA to store time with millisecond support.
2. A new base c-tree type, *CTTIMEMS*, was added to hold a time with millisecond format. *CTTIME* and *CTTIMEMS* are not comparable without transformation. *CTTIME* is a LONG value holding the number of seconds since midnight, *CTTIMEMS* is a LONG value holding the number of milliseconds since midnight.
3. New FairCom DB API functions were added to set and get files as CTTIMEMS, specifically **ctdbGetFieldAsTimeMsec** and **ctdbSetFieldAsTimeMsec**.

   Calling **ctdbSetFieldAsTimeMsec** against a *CT_TIME* field would cause a loss of milliseconds. If this attempted, the error **CTDBRET_MSEC_NOTSUPPORTED** (4147) is returned.
4. FairCom DB API and c-treeACE SQL now support time and timestamp with millisecond precision on newly created tables.
5. New functions were added to FairCom DB API time with millisecond manipulation functions: **ctdbTimePackMsec** and **ctdbTimeUnpackMsec.**

Searching an index based on *CT_TIME* is problematic when milliseconds are significant since the index information does not contain milliseconds and, as we do with filter, we cannot automatically add milliseconds before doing the search. SQL will error out when performing index searches on *CT_TIME* columns.

Note: This support requires new extended data types (page 252, /doc/v11ace/68793.htm) in c-treeACE V11 forward.

Several functions were added to FairCom DB API related to TIMESTAMP (CTDATETIME):

CTDATETIME <-> CTTIMEMS conversions:

```
ctdbDateTimeSetTimeMsec()
ctdbDateTimeGetTimeMsec()
```

Millisecond DateTime handling:

```
ctdbDateTimePackMsec()
ctdbDateTimeUnpackMsec()
```

Timestamps with milliseconds are supported on existing timestamp data with no conversion. A minor behavior change should be noted:

Prior to this release, a query similar to the example shown below would have ignored the milliseconds portion, returning records matching '11/11/2015 12:00:15.000':

```
SELECT * FROM mytable WHERE mytimestamp = '11/11/2015 12:00:15.998'
```

With the changes described above, the milliseconds are now significant.

13.2 Table Lock Support

In V11 and later, c-treeACE supports file-level c-tree data file locks, or more commonly referred to as "table locks." An application can request a file lock to prevent other connections from reading from and/or writing to the file.

- A file read lock (table read lock or shared table lock) allows other connections to acquire read locks on records in the file but not write locks.
- A file write lock (table write lock or exclusive table lock) prevents other connections from acquiring read and write locks on records in the file.

These locks can be invoked using standard SQL commands or from any of the supported c-treeACE interfaces using commands similar to those shown later in this section.

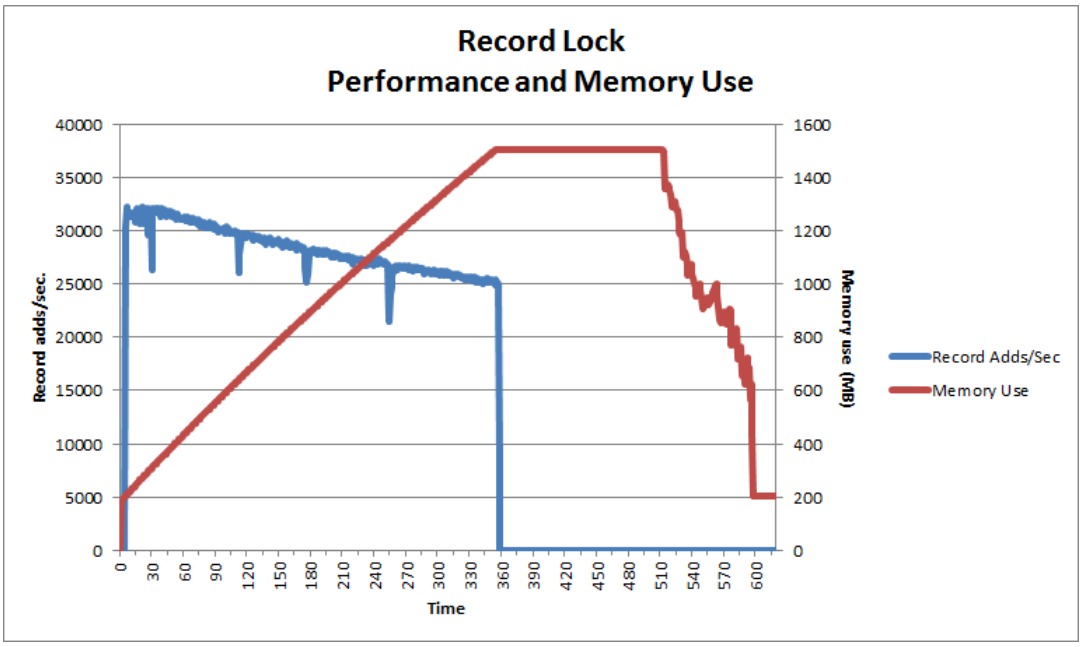
Table locks reduce resource usage with large sequences of actions on a table. A bulk add operation, for example, generates many lock requests. With a table lock the table is in an exclusive open state, thereby avoiding the necessity of managing a large number of new lock requests. This benefits both memory resource usage, as well as great gains in performance as overhead of lock management is greatly reduced.

Performance

The chart below shows the results of a test of the table lock. A C# stored procedure is adding 10 million records to a c-tree data file. The data file is a non-transaction file and has no index (to help isolate the effect of record locking).

The test was run twice: first with individual record locks and then with a table lock. In the chart below, the blue line indicates performance (records added per second) and the red line indicates memory use.

The chart below shows the results for the individual record lock case: the record insert performance (blue line) decreases over time, and memory use (red line) increases over time. And when the adds are finished and the commit happens, a noticeable time is spent freeing those locks. Insert time: **356 sec**.



The chart below shows the results for the table lock case: *the record insert performance (blue line) is higher than with individual record locks and stays at the same level of performance throughout*. The memory use (red line) initially increases by a small amount and then stays at that level. Insert time: **284 sec** *(20% faster)*.

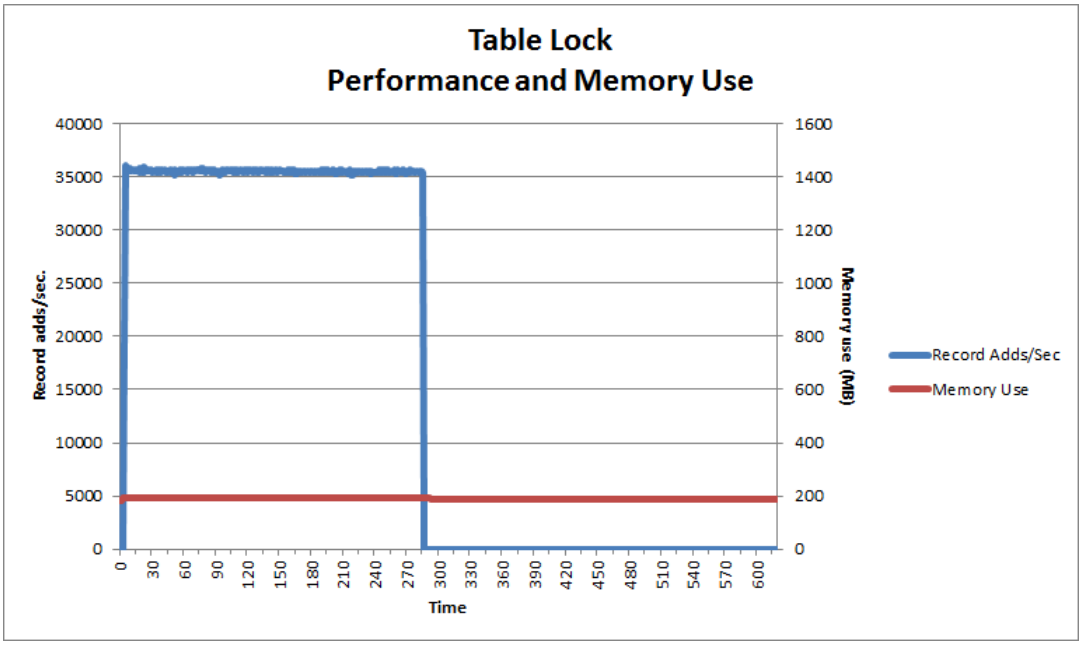


Table Locks with c-treeACE SQL

In applications where a large number of rows will be accessed for either reading or modifying, SQL provides an explicit locking construct for locking all the rows of a table. The `LOCK TABLE` statement explicitly locks a table in either `SHARE` or `EXCLUSIVE` mode.

The following example shows how to acquire a lock in the `EXCLUSIVE` mode for a table called `customer` from an ESQL command line:

```
EXEC SQL
     LOCK TABLE customer IN EXCLUSIVE MODE;
```

The above statement will prevent other transactions from either reading or modifying the table customer until the transaction either commits or performs a rollback.

The following example shows acquiring lock in the `SHARE` mode for a table called `orders`:

```
EXEC SQL
     LOCK TABLE orders IN SHARE MODE;
```

The above statement will prevent other transactions from modifying the `orders` table until the transaction either commits or performs a rollback.

Table Locks with c-tree API Functions

To acquire a file (table) lock using c-treeACE, call **LOKREC()** (https://docs.faircom.com/doc**/ctreeplus/lockctdata.htm**) with a new *ctLOCK_FILE* mode as follows:

```
LOKREC(datno, ctLOCK_FILE, lockmode);
```

See **Table Lock Mode for LOKREC** (page 162) for details.

Table Lock Behavior with Transactions

If a table write lock is in effect when a record is updated in a transaction, the table write lock cannot be removed until the transaction commits or aborts. Note that the call to free the table write lock returns success (NO_ERROR) and *sysiocod* is set to **UDLK_TRN (-3)** in this situation, indicating that the table write lock was not released.

13.3 IPv6 Support

Internet Protocol (IP) IPv6 format is available for V11 and later c-treeACE servers.

IPv6 greatly expands the number of available IP addresses over the previous IPv4 standard, which used the familiar quad dotted format (four decimal numbers, from 0 through 255, separated by dots), allowing 4.3 billion addresses. IPv6 uses eight groups of four hex digits each, for 2^{128} addresses. Although proposed in 1998, only a small percentage of machines are currently using IPv6. However, IPv6 adoption is quickly expected as the number of IPv4 addresses is at, or very near, exhaustion, especially for external internet addressing. Although, IPv4 remains efficient for internal networks.

- IPv6 support is not available for QNX and SCO.

Server Configuration Options

This support is affected by placing the following configuration option in *ctsrvr.cfg*:

- `COMM_PROTOCOL F_TCPIPV6` - Starts a new listener thread on an IPv6 socket.

To support both IPv4 and IPv6 in the same c-treeACE Server, list the following keywords:

- `COMM_PROTOCOL F_TCPIP`
- `COMM_PROTOCOL F_TCPIPV6`

Server keyword `SQL_OPTION NO_IPV6` can be added to *ctsrvr.cfg* to accept only IPv4 connections. This is not generally recommended and is more likely to cause connection issues than to solve them.

Note: The `COMM_PROTOCOL F_TCPIPV6` keyword is commented out in the default *ctsrvr.cfg* file. Be sure to remove the comment symbol (the semicolon - `;`) before trying to use this new support.

An environment variable can be used for native clients to request only IPv4 address: *CTSQL_IPV4_ONLY*. Setting this variable to any value in the environment will effectively disable IPv6 connection attempts from the client. This may be needed on networks where both IPv4 and IPv6 are enabled, but the c-tree SQL server does not accept IPv6 connections.

Native SQL clients on Windows will attempt to connect to the first address (IPv4 or IPv6) resolved for the host.

Notes:

Server keyword `SQL_OPTION NO_IPV6` can be added to *ctsrvr.cfg* to accept only IPv4 connections. This is not generally recommended and is more likely to cause connection issues than to solve them.

Native SQL clients on Windows will attempt to connect to the first address (IPv4 or IPv6) resolved for the host.

Java and ADO.NET SQL clients currently support only IPv4. Explicit IPv6 addresses in client connection strings are not currently supported.

Default Protocol Override

An application can override the default protocol by specifying the protocol in the connection string with the following syntax:

```
FAIRCOMS@myhost^TCPIP
```

or

```
FAIRCOMS@myhost^TCPIPV6
```

Building IPV6 Client Applications

By default, c-treeACE libraries and utilities are built with IPV4 support only. For IPV6 support, build your own c-treeACE libraries and utilities using **mtmake.exe** (**mtmake** on Linux) or **mtpro.exe** (the Windows-only GUI version) located in the "*pro*" directory (by default: *\FairCom\V*\win32\pro*). On the client side, when IPv6 is selected in **mtmake**, both V4 and V6 stacks are included.

Note: As IPv6 adoption grows, default mtclient libraries will likely be enabled with this support in the near future.

If a numeric address is used, or the protocol is specified with the `^` syntax, we use the desired protocol. Otherwise we first attempt an IPv6 name resolution and connection. If that fails we attempt an IPv4 name resolution and connection.

Note: On Windows, IPv6 requires *winsock2.h* and linking with *ws2_32.lib*. For applications, this will mean compilation errors if *windows.h* is included before *ctreep.h* or *winsock2.h*.

GetServerBroadcast Function

IPv6-enabled clients can use the new function:

```
NINT GetServerBroadcast(pTEXT buffer, NINT bufsiz, UCOUNT port, LONG sec, NINT protocol)
```

This function provides functionality equivalent to the IPv4-only function **GetServerInfoXtd()**.

The protocol must be either of the constants `lcyNTREE_TCP` or `lcyNTREE_TCPIPV6`.

Note: With node-based licensing, if a single client machine (besides a local connection) connects to the server using both IPv4 and IPv6 addresses, it will count as 2 nodes.

13.4 ReFS

FairCom has an ongoing commitment to provide our customers with the latest technologies and most up-to-date feature set. As a Microsoft Silver Application Development Competency Partner, we have paid close attention to supporting the latest features on this operating system. Microsoft Resilient File System, ReFS, is available with Microsoft Server 2012 and above as well as Windows 8.1. Their new file system provides a rich and robust feature set focused on Integrity, Availability, Scalability, App Compatibility, and Proactive Error Identification.

Our architecture allows us to fully utilize the key advantages of this advanced file system. FairCom users can experience clear performance gains with ReFS. Coupled with FairCom's

advanced engineering-level technology, ReFS can empower customers with a cost-effective, highly-available, high-performance database solution.

13.5 Prevent ISAM Index Key Value Updates

Beginning with V11, it is possible to create an index that does not allow an ISAM update to change the key values. On such an index, an ISAM record add can add a new key value and an ISAM record delete can delete a key value, but an ISAM record update cannot change a key value. (Note the special case for a variable-length record: if an ISAM update causes the record to move, the key is allowed to change its record offset from the old offset to the new offset; but the key value itself is not allowed to change.)

To create an index with this feature enabled, OR *KTYP_NOISMUPD* into the key type (*ikeytyp*) field of the *IIDX* structure for that index file before calling **CREIFIL()** to create the index.

PUTIFIL() can be used to turn this bit on or off. Changes take effect immediately.

An ISAM record update that attempts to change a key value for an index with this property enabled fails with error **UKEY_ERR** (1001).

13.6 Automatic Directory Creation for New Files

When creating a c-tree data or index file,c-treeACE now supports automatically creating directories specified in the filename when they do not exist. The AUTO_MKDIR configuration option enables this feature. Specify `AUTO_MKDIR YES` in *ctsrvr.cfg* to enable this feature. This feature is disabled by default.

See Also

- New Xtd8 File Mode to Automatically Create Directories (page 164)

13.7 Extended FairCom DB API Default Field Value Support

In V11 and later, it is possible to set default field values using functions such as SYSDATE, SYSDATETIME, etc. Previously, FairCom DB API only supported defining default values for fields using literal values. This improves FairCom DB API compliance with SQL, where it is possible to set default field values with literals or with functions.

A new enum, `CTDEF_TYPE`, has been added to use with these new functions:

```
CTDBRET ctdbDECL ctdbSetFieldDefaultValueType(CTHANDLE Handle, CTDEF_TYPE def_type);
```

Sets the default value type.

```
CTDEF_TYPE ctdbDECL ctdbGetFieldDefaultValueType(CTHANDLE Handle);
```

Gets the default value type.

Valid def_types are:

`CTDEF_LITERAL`: the value is a literal

`CTDEF_USER`: the current* user name

`CTDEF_NULL`: null

`CTDEF_SYSDATE`: the current* date

`CTDEF_SYSTIME`: the current* time

`CTDEF_SYSTIMESTAMP`: the current* timestamp

* “current” means at the time the default gets evaluated to be stored in the field.

To set the default value as a literal and automatically set its type to `CTDEF_LITERAL`, use a call to **ctdbSetFieldDefaultValue**.

To set the default value to any of the supported SQL functions, use a call to **ctdbSetFieldDefaultValueType** and pass the desired type, e.g. `CTDEF_SYSDATE`.

Default Values with ALTER TABLE

ctdbAlterTable has been enhanced to properly set the default value in new fields when doing a full rebuild.

ctdbAlterTable logic has been slightly changed so that the default value gets applied only to the fields just added, not to existing fields for which it may have been added or changed. Before this modification, existing fields could unintentionally change their values from NULL to the default value.

13.8 Retrieve Current Server Date and Time

In V11 and later, two modes were added to the c-treeACE **GETNAM()** API function (full name **GetSymbolicNames**) to retrieve the current server-side date and time:

- *CURTIME* - Retrieve the current server time expressed as the number of seconds since 01/01/1970. This was implemented for a 4-byte buffer.
- *CURTIMETM* - Retrieve the current server time in the local time (timezone, daylight, etc). This mode retrieves a buffer of ctTM time, which is similar to the regular "struct tm" type.

A new FairCom DB API function was added in parallel:

```
CTDATETIME ctdbServerDateTime(CTHANDLE hSession)
```

where *hSession* is a session handle.

13.9 User-Defined Function for Conditional Expressions

New "User-Defined Function" (UDF) support for conditional expressions is available. A new function pointer type called *ctdbUDFFunc* enables this support.

```
VOID (ctdbDECL* ctdbUDFFunc)( ppVOID args, pVOID retval );
```

where:

- *args* is a list of argument pointer.
- *retval* is a return value pointer.

A new FairCom DB API API is provided to add a UDF to the given session handle:

```
CTDBRET ctdbAddUserDefFunc(CTHANDLE Handle, pTEXT funcName, ctdbUDFFunc funcPointer, CTDBTYPE funcRetType,
pCTDBTYPE funcArgTypes, COUNT funcArgCount);
```

where:

- *Handle* is a session handle.
- *funcName* is the UDF name.
- *funcPointer* is the UDF pointer.
- *funcRetType* is the UDF return type.
- *funcArgTypes* is the UDF argument type list.
- *funcArgCount* is the UDF argument list count.

If a function name is not found while parsing an expression, it will be searched on the UDF list by name and it will be executed.

Example:

```
/* UDF body for a function that inverts a string: "TEST" -> "TSET" */
VOID udftest( ppVOID args, pVOID retval )
{
    pTEXT arg1  = (pTEXT)args[0];
    pTEXT ret   = (pTEXT)retval;
    COUNT i, arglen = (COUNT)strlen( arg1 );
    for( i = 0; i < arglen; i++ )
        ret[i] = arg1[arglen - i - 1];
    ret[arglen] = 0;
}

/* Adding UDF for the session handle */
CTDBTYPE arglist[1] = { CT_STRING };
ctdbAddUserDefFunc( hSession, "udftest", udftest, CT_STRING, arglist, 1 );

/* Parser expression using UDF for the record handle */
ctdbParserRecordExpr(hRecord, "udftest( cm_custname )", &exprType, &exprIdx);"
```

When FairCom DB API is active in the library, it is now possible to add a UDF to c-tree's expression evaluator by defining a UDF list.

Using ISAM/low-level functions to define a UDF list for the current c-tree instance, the list must be initialized with a call to:

```
udfInit()
```

Each UDF can be added by calling:

```
COUNT udfAdd( pTEXT name, ctdbUDFFunc funcPrt, CTDBTYPE retType, pCTDBTYPE argTypeList, COUNT argListCount
pinHan )
```

Before exiting the c-tree environment the list can be released by calling:

```
udfTerm()
```

The udfAdd function takes a function pointer having as type ctdbUDFFunc as follows:

```
VOID (ctdbDECL* ctdbUDFFunc)( ppVOID args, pVOID retval );
```

where:

- *args* is a list of argument pointer.
- *retval* is a return value pointer.

When using FairCom DB API, it is possible to add a new UDF for the current session by calling:

```
CTDBRET ctdbAddUserDefFunc(CTHANDLE Handle, pTEXT funcName, ctdbUDFFunc funcPointer, CTDBTYPE funcRetType,
pCTDBTYPE funcArgTypes, COUNT funcArgCount);
```

where:

- *Handle* is a session handle.
- *funcName* is the UDF name.
- *funcPointer* is the UDF pointer.
- *funcRetType* is the UDF return type.
- *funcArgTypes* is the UDF argument type list.
- *funcArgCount* is the UDF argument list count.

If a function name is not found while parsing an expression, it will be searched on the UDF list by name and it will be executed.

Example:

```
/* UDF body for a function that inverts a string: "TEST" -> "TSET" */
VOID udftest( ppVOID args, pVOID retval )
{
    pTEXT arg1  = (pTEXT)args[0];
    pTEXT ret   = (pTEXT)retval;
    COUNT i, arglen = (COUNT)strlen( arg1 );
    for( i = 0; i < arglen; i++ )
        ret[i] = arg1[arglen - i - 1];
    ret[arglen] = 0;
}

/* Adding UDF for the session handle */
CTDBTYPE arglist[1] = { CT_STRING };
ctdbAddUserDefFunc( hSession, "udftest", udftest, CT_STRING, arglist, 1 );

/* Parser expression using UDF for the record handle */
ctdbParseRecordExpr(hRecord, "udftest( cm_custname )", &exprType, &exprIdx);"
```

13.10 System Group Assignment of Unix/Linux Shared Memory resources

On Unix/Linux systems, a user can belong to more than one group of which one group is the primary group, and all other groups are secondary groups. When the `SHMEM_PERMISSIONS` option is used to only enable user and group permissions on shared memory resources, the resources created for shared memory connections (files, semaphores, shared memory regions) are assigned with the user's current primary group by default.

To address this situation, a new configuration option, `SHMEM_GROUP`, has been added preventing a user account that shares a secondary group with the user account under which the FairCom Server process is running failing to connect using shared memory.

This option causes the FairCom Server to assign group membership to the specified group. This option applies to the resources for both the ISAM and the SQL shared memory protocol.

As an example, consider two user accounts:

- `user1` - belongs to groups `group1` and `group2`
- `user2` - belongs to group `group2`

If the `user1` account runs the FairCom Server with `SHMEM_PERMISSIONS 660` in *ctsrvr.cfg*, a client program run by the `user2` account will fail to connect using shared memory.

To allow the client program run by `user2` to connect, add the following configuration option to *ctsrvr.cfg* and restart FairCom Server:

```
SHMEM_GROUP group2
```

This causes the shared memory resources to be assigned to group `group2`, which allows the `user2` client program to connect.

13.11 Specify Shared Memory Keys on Unix

When more than one FairCom Server was run on a Unix system, the shared memory keys used by different servers could have the same value, which prevented connections to the servers. In addition, it was possible for unrelated applications to collide with default keys generated by c-treeACE servers.

To address this key collision, it is now possible for an administrator to specify specific shared memory keys for ISAM and SQL shared memory communication protocols ensuring the keys do not match existing keys already in use on the system.

New FairCom Server configuration options are available to directly specify a shared memory key. SQL and ISAM each require separate shared memory support

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

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%
```

Compatibility Notes:

When these configuration options are *not* used, FairCom Server uses the old method of assigning shared memory keys so its shared memory communication protocol is compatible with old clients. FairCom Server now writes the shared memory key to the shared memory resource file and new clients know to read the shared memory key from this file. If a new client finds no shared memory key in the file, it uses the old method to assign a shared memory key so it is compatible with an old server.

The shared memory resource file is named */tmp/ctreedbs/<server_name>* for ISAM, and */tmp/ctreedbs/CTSQL_<sql_port>* for SQL, where */tmp/ctreedbs* is the default shared memory directory. It can be changed using the `SHMEM_DIRECTORY` configuration option.

An old client will not be able to connect to a new server using shared memory if the server uses the `SHMEM_KEY_ISAM` or `SHMEM_KEY_SQL` configuration option to specify a shared memory key that differs from the shared memory key that the old method would generate.

13.12 FairCom Server - Configuration option to disable delete node thread

FairCom Server's delete-node thread prunes empty nodes from c-tree index files in the background. This maintains index key data densely packed for optimal performance. This activity requires directly opening the index file by the delete-node thread, which happens during idle times when it may be expected external applications can access the file. Having the file open at the time by the delete-node thread prevents external file open.

In exceptional cases, this behavior may not be desirable as external processes expect complete access to the file when it is no longer in use by the application. For example, processes may wish to immediately copy the file for other external processing.

In V11 and later, FairCom Server supports a configuration option to disable the internal delete-node thread. The option `COMPATIBILITY NO_DELNOD_QUEUE` disables the delete-node thread and disables the writing of entries to the delete-node queue.

Caution: This option should only be used in special situations in which you absolutely require an external process access to closed files, which is discouraged if at all possible while the server is operational.

Operating without pruning empty nodes from indexes can potentially diminish performance of certain index search operations which may non-optimally traverse many empty leaf nodes. This is especially true for applications which heavily delete records from the database.

13.13 Monitor c-treeACE Memory Use and Suballocator List Allocation Call Stacks

Support has been added for monitoring FairCom Server memory use and collecting allocation call stacks for each suballocator list. The ability to monitor FairCom Server's memory use has been enhanced in the following ways:

- It now tracks the number and byte count of allocations that do not go through c-treeACE's memory suballocator.
- FairCom Server now makes its memory suballocator usage figures available to monitoring tools. The **ctstat** utility's *-ml* option can be used to display current memory allocation figures. Example:

```
# ctstat -ml -t -i 2 -h 1 -s FAIRCOMS

Thu May 01 14:31:22 2014
  name      allocated         in use       pct
KLNTYP              0              0      0.00%
PI1TYP          32768           1704      0.01%
PI2TYP          32768           6800      0.01%
PI4TYP        1376256        1224360      0.31%
PI8TYP        1245184        1155592      0.28%
PIwTYP          32768          19008      0.01%
PIxTYP          32768          16120      0.01%
PIyTYP          49152          25800      0.01%
PIzTYP          32768              0      0.01%
PIaTYP          82560          69768      0.02%
PIbTYP         296064         270600      0.07%
SHDTYP              0              0      0.00%
BATTYP              0              0      0.00%
ILKTYP          16384              0      0.00%
FNMTYP              0              0      0.00%
COMTYP          16384              0      0.00%
ABTTYP              0              0      0.00%
LOKTYP          16384              0      0.00%
DCMTYP              0              0      0.00%
IXCTYP       32129024       29018360      7.24%
DTCTYP        6782976        6547464      1.53%
CTCTYP          81920          70000      0.02%
IXBTYP      180854968      180854968     40.74%
DTBTYP      203169792      203169792     45.76%
MBATYP       17690815       17690815      3.98%
TOTAL:      443971703      440141151
```

Note: If memory allocation call stack is enabled for a suballocator list, the name of the suballocator list is followed by an asterisk in this output (for example, as MBATYP*).

Just as the **ctstat** utility does, a FairCom Server client can read the memory use figures by calling the **ctSNAPSHOT()** API function with the new mode *ctPSSmemAlloc*. This mode returns the memory use figures in a new structure named ctGMMS. See *ctstat.c* for an example of this **ctSNAPSHOT()** call.

- On Windows and Linux systems, FairCom Server supports collection of call stacks for memory allocations. This support, which existed prior to this revision, is enabled using the configuration option `DIAGNOSTICS MEMTRACK`. Now FairCom Server allows allocation call stack collection to be dynamically enabled or disabled for specific memory suballocator lists, provided that `DIAGNOSTICS MEMTRACK` was specified in the configuration file. The **ctMEMSTAT()** API function is used to change these settings, and the **ctstat** utility's *-mt* option provides a convenient way to use this function.

Only a member of the ADMIN group is allowed to change memory allocation settings.

Memory Suballocator List Descriptions

- **KLNTYP** - key buffer, size of the largest supported maximum key value (MAXLEN = 1024)
- **PI1TYP** - allocations of size > 0 and <= PI_UNIT (PI_UNIT = 16)

- **PI2TYP** - allocations of size > PI_UNIT and <= 2 * PI_UNIT
- **PI4TYP** - allocations of size > 2 * PI_UNIT and <= 4 * PI_UNIT
- **PI8TYP** - allocations of size > 4 * PI_UNIT and <= 8 * PI_UNIT
- **PIwTYP** - allocations of size > 8 * PI_UNIT and <= 16 * PI_UNIT
- **PIxTYP** - allocations of size > 16 * PI_UNIT and <= 32 * PI_UNIT
- **PIyTYP** - allocations of size > 32 * PI_UNIT and <= 64 * PI_UNIT
- **PIzTYP** - allocations of size > 64 * PI_UNIT and <= 128 * PI_UNIT
- **PIaTYP** - allocations of size > 128 * PI_UNIT and <= 256 * PI_UNIT
- **PIbTYP** - allocations of size > 256 * PI_UNIT and <= 512 * PI_UNIT
- **SHDTYP** - preimage space list entry, size of SHADLST structure (the variable-length contents are allocated separately if larger than ctSHADOWlen)
- **BATTYP** - batch retrieval list entry, size of BATLST structure
- **ILKTYP** - user lock table list entry, size of LOKS structure
- **FNMTYP** - file name, size of the largest supported file name (MAX_NAME = 255)
- **COMTYP** - commit node list entry, size of COMLST structure
- **ABTTYP** - abort node list entry, size of ABTLST structure
- **LOKTYP** - system lock table list entry, size of RECLOK structure
- **DCMTYP** - commit data list entry, size of COMLST structure
- **IXCTYP** - index cache page control structure, size of TREEBUFF structure (allocated at startup and does not change after that)
- **DTCTYP** - data cache page control structure, size of DATBUF structure (allocated at startup and does not change after that)
- **CTCTYP** - system file control block entry, size of CTFILE structure
- **IXBTYP** - index cache page buffer, size of PAGE_SIZE + MAXLEN + 2 * sizeof(ctRECPT) (allocated at startup and does not change after that)
- **DTBTYP** - data cache page buffer, size of PAGE_SIZE (allocated at startup and does not change after that)
- **MBATYP** - allocation size > 512 * PI_UNIT

Examples:

1. Enable memory allocation call stack collection for all suballocator lists:
   ```
   ctstat -mt +ALL -u ADMIN -p ADMIN -s FAIRCOMS
   ```
2. Enable memory allocation call stack collection for only the MBATYP and LOKTYP suballocator lists:
   ```
   ctstat -mt +MBATYP,+LOKTYP -u ADMIN -p ADMIN -s FAIRCOMS
   ```
3. Disable memory allocation call stack collection for all suballocator lists:
   ```
   ctstat -mt -ALL -u ADMIN -p ADMIN -s FAIRCOMS
   ```

As before, the current memory allocations can be logged to a file using the **ctMEMSTAT()** function, as used by the **ctstat** utility's *-ma* option:

```
ctstat -ma mem.log -i 1 1 -u ADMIN -p ADMIN -s FAIRCOMS
```

Compatibility Notes:

1. The use of the existing and new memory monitoring options is now restricted to members of the ADMIN group.
2. When memory allocation call stack tracking is supported on a per-suballocator list basic, the `DIAGNOSTICS MEMTRACK` option must still be specified in *ctsrvr.cfg* to support collecting memory allocation call stacks, but collection of allocation call stacks is initially disabled for all suballocator lists. An administrator must use the **ctstat** utility's *-mt* option as shown above to enable collection of memory allocation call stacks for the desired suballocator lists.

   If troubleshooting unexpected memory growth, first use **ctstat** *-ml* to monitor memory use by suballocator list. Then enable memory allocation call stack collection just for the lists that show the unexpected growth. This approach can reduce the overhead of the memory allocation call stack collection and can simplify analysis of the unexpected memory growth.

Memory Tracking with c-treeACE on Linux

In V11 and later, the memory allocation tracking feature of FairCom Server is supported on Linux systems. To use this feature, add `DIAGNOSTICS MEMTRACK` to *ctsrvr.cfg*.

The `MEMTRACK` keyword allows the **ctstat** utility to be used to log current memory allocations to a file. The following parameters can be used with **ctstat**:

- *-mf logfile* - Log all memory allocations to the specified file
- *-ma logfile* - Log aggregate memory allocations to the specified file
- *-mr min,max* - Log only memory allocations in the range min,max
- *-ms* - Output memory allocation statistics

Examples

```
C:\>ctstat -ms -h 10 -s FAIRCOMS
      memseq      memalc
        1267         992
        1289         997
        1289         997
        1289         997
```

To log all memory allocations (with each allocation listed separately) to the file *memfull.log* in the FairCom Server's working directory:

```
C:\>ctstat -mf memfull.log -i 1 1 -s FAIRCOMS
```

To log all memory allocations (with allocations having the same call stack listed only once each) to the file *memaggr.log* in the FairCom Server's working directory:

```
C:\>ctstat -ma memaggr.log -i 1 1 -s FAIRCOMS
```

To log all memory allocations that have sequence numbers between 1900 and 2000 to the file *memaggr.log* in the FairCom Server's working directory:

```
C:\>ctstat -ma memaggr.log -mr 1900,2000 -i 1 1 -s FAIRCOMS
```

13.14 Unicode Support

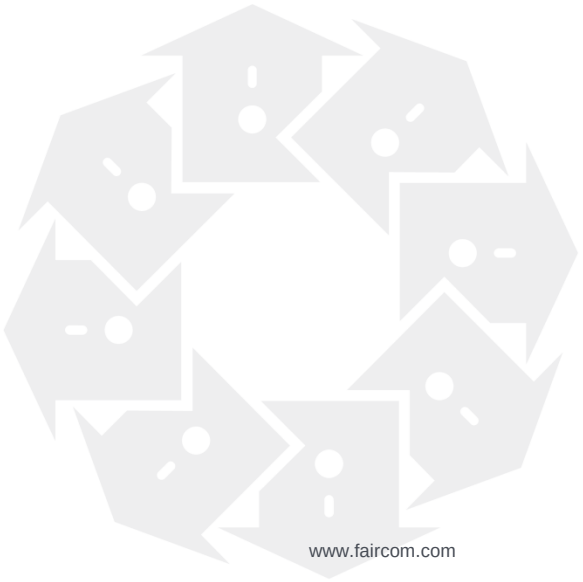
All versions of c-treeACE now support Unicode.

13.15 Latest Microsoft Visual Studio 2015 Support

c-treeACE Professional development environment now supports Microsoft Visual Studio 2015 projects and solutions.

14. Additional File Management Options

c-treeACE V11 see many advances in the area of file management. This section lists those changes.



14.1 Copy Files Between c-treeACE Servers

c-treeACE users have long enjoyed the ease of managing their physical data and index files alongside their applications. Easily moving and copying files between servers is a consistent need required by our customers, especially as they transition applications to centrally processed c-treeACE server technology. A file copy function makes this a seamless effort directly from the application layer. File transfer is now done server-to-server in addition to a client/server transfer feature introduced in V10.

The function **ctCopyFile()** (https://docs.faircom.com/doc**/ctreeplus/ctcopyfile.htm**) copies a c-tree data or index file, or a c-tree data file and its associated index files. The caller specifies the source and destination file names and options that affect the behavior of the file copy. c-tree copies the files synchronously. That is, the copy call does not return until the file copy is completed.

To copy a file, a caller must be authorized by c-tree (including the file system) to open a file for read access. ADMIN user is always granted read permission for all files. (c-tree file permissions are only supported in the client/server model and not with standalone usage.)

c-tree attempts to open the file in exclusive mode, while still allowing other connections to open the file for read access. If the file is already open, c-tree flushes updates to disk and sets the update flag to zero to indicate that the file is in a clean, consistent state. If c-tree cannot open the file in exclusive mode, the copy operation fails.

Note: Files with an *.FCS* extension (such as *FAIRCOM.FCS*, *SYSLOGDT.FCS*, *L0000001.FCS*, *SNAPSHOT.FCS*, etc.) are considered to be a c-tree internal system housekeeping files. The file copy function does not allow copying from or to c-tree internal system files. An attempt to copy such a file fails with error code 1013 (**SCPY_ERR**).

Copying c-tree files

When copying a c-tree file, c-tree assigns a new file ID to the copy of the file so that the file ID does not match the original file's ID, and c-tree sets the data and index file and directory names in the data file's *IFIL* resource to the names of the new copies of the files. Other than those differences, the copied file is identical to the original file: it contains the same active and deleted data records or index nodes and key values, resources, and has the same properties as the original file, including:

- security (file owner, file group, file permission mask, file password)
- data encryption
- data compression
- transaction control

The caller indicates what action to take when a file already exists with the same name as the specified destination file. By default, the file copy fails in this situation, but the caller can specify that the existing file should be deleted instead.

When copying multiple files (for example when copying a data file and its associated index files), if an error occurs c-tree stops the file copy operation and deletes all files that were successfully copied. If desired, the caller can specify an option to keep all the successfully copied files in case of an error.

An optional callback mechanism is available for supporting user feedback progress bars based on percentage of file copy complete.

See also

- **ctCopyFile (https://docs.faircom.com/doc/ctreeplus/ctcopyfile.htm)** - API and usage details
- COMPATIBILITY COPY_FILE_OPEN_SHARED (https://docs.faircom.com/doc/ctserver/compatibility-copy-file-open-shared-config.htm)
- **ctfcpAllocateHandle** (page 186)
- **ctfcpFreeHandle** (page 184)
- **ctfcpAddFileCopyOperation** (page 185)
- **ctfcpRemoveFileCopyOperation** (page 187)
- **ctfcpSetErrorBuffer** (page 188)
- **ctfcpGetErrorBuffer** (page 189)
- **ctfcpSetCallback**
- **ctfcpSetCopyOptions** (page 191)
- **ctfcpSetCopyFilenames** (page 192)
- **ctfcpSetCopyFilePassword** (page 193)
- **ctfcpSetServerParameters** (page 194)
- **ctfcpCopyFile** (page 195)

14.2 File Copy Wrapper API Functions

Due to the large number of parameters involved in server-to-server file copy operations, a set of wrapper functions has been added, allowing easier usage of file copy functionality.

Review API details in the Interfaces (page 158) section

- **ctfcpAllocateHandle()** (page 186)
- **ctfcpFreeHandle()** (page 184)
- **ctfcpAddFileCopyOperation()** (page 185)
- **ctfcpRemoveFileCopyOperation()** (page 187)
- **ctfcpSetErrorBuffer()** (page 188)
- **ctfcpGetErrorBuffer()** (page 189)
- **ctfcpSetCallback()**
- **ctfcpSetCopyOptions()** (page 191)

- **ctfcpSetCopyFilenames()** (page 192)
- **ctfcpSetCopyFilePassword()** (page 193)
- **ctfcpSetServerParameters()** (page 194)
- **ctfcpCopyFile()** (page 195)

14.3 Verify Data and Index File Integrity

Situations arise where it can be desirable to perform a full data and index file integrity check. After a total system outage for example. Or a major application upgrade might want to confirm that all data remains intact. A new verification utility function has been introduced providing this ability.

ctVerifyFile() (, **/doc/ctreeplus/ctverifyfile.htm**), will verify c-treeACE data and index integrity in . It is used by the new included **ctvfyfil** utility.

In addition, a file verification utility has been added based on this API function. This makes it easy to script integrity checks as needed.

The ctvfyfil Utility

The **ctvfyfil** utility calls the **ctVerifyFile()** (, **/doc/ctreeplus/ctverifyfile.htm**) function. (See function for details.) The utility can be run in standalone and in client/server mode.

Usage

```
ctvfyfil [<option> ...] <file name>
```

where *<option>* is one or more of the following:

- *<page size>* - Use the specified page size
- *-chkdat* - Read data file only (default)
- *-chkdatkeys* - Read data file and check keys in index
- *-chkdeletedspace* - Check deleted space
- *-chkidx* - Read index file
- *-chkidxrec* - Read index file and check records in data file
- *-chkidxint* - Additional index checks
- *-excl* - Open the file in exclusive mode
- *-int* - Interactive mode: stop on each error
- *-index=<N>* - Check only the *N*th index (N=1,2,...)
- *-local* - causes the utility to use a standalone (local side) connection instead of a client connection when linked with a LOCLIB library.

The example below shows the utility run on a file called *mark.dat* with the page size set to 8192:

```
ctvfyfil -8192 mark.dat
```

See ctVerifyFile() (, /doc/ctreeplus/ctverifyfile.htm) API for complete details and usage.

14.4 Delete Node and Space Reclamation Threads no Longer Preclude File Access

Logic was added in V11 to control file access by the Delete Node and Space Reclamation threads. These changes are available for c-treeACE when atomic operation support is enabled. This feature is intended to avoid a client being unable to open a c-tree file in exclusive mode because an internal thread has the file open.

14.5 Toggle Serial Segment (SRLSEG) Support for Files

c-treeACE V11.0 and later supports disabling/enabling *SRLSEG* with a **PUTIFIL()** call and during compact and rebuild when the *updateIFIL* option is specified. A file that contains an extended header now supports turning the *SRLSEG* or *SCHSRL* segment mode on or off by:

1. A call to **PUTIFIL()**.
   or
2. A call to the rebuild and compact function with the *updateIFIL* option specified in the IFIL's *tfilno* field.

Note: A call to rebuild or compact will fail with error **IAIX_ERR** (608) if the key segment definitions specified by the caller change the serial number attributes (either turning serial number on or off or changing its location in the record) when the *updateIFIL* option is not specified in the *tfilno* field of the IFIL structure.

14.6 New File Descriptor Operational Parameters

A server configuration keyword, FILES, sets the maximum number of files to be open at one time.

There is no effective limit to the number of files supported by the FairCom Server, except for any limits imposed by the operating system and available system memory.

Note: The number of file descriptors set by the `FILES` keyword may need to be considerably greater than the number of open data files. Each index, whether or not in a separate file, counts toward this total. For example, a host index file that supports three different keys (i.e., contains three separate index members) counts as three files toward the `FILES` total. In addition, each member of a superfile is counted toward the total set by the `FILES` keyword.

Unix/Linux Considerations

The Unix and Linux operating systems place a limit on the number of file descriptors that can be in use at one time. The number of file descriptors required by c-treeACE is determined as follows:

- Each open file consumes 1 file descriptor (this may be somewhat lower than the setting of the `FILES` keyword because individual superfile members and members of a host index do not consume file descriptors).

- Each TCP/IP socket consumes a file descriptor (this corresponds to the `CONNECTIONS` keyword).
- c-treeACE reserves a few file descriptors in addition to those used for data files, index files, and TCP/IP connections.

Based on the above considerations, the system should be configured to allow a number somewhat greater than the number of **files + connections**.

The file descriptor limit is typically set by the **limit** or **ulimit** command, which may require superuser access, or by the hard limit set in a system configuration file such as */etc/security/limits.conf* on Linux. The specifics vary by system, so consult the documentation for your Unix or Linux system. (Unix and Linux define both hard limits and soft limits. A soft limit can be changed by a process at any time, but it cannot exceed the hard limit. The hard limit is of interest for this discussion.)

Note: When the file descriptor limit for FairCom Server (set by the operating system) is set too low, server startup fails with error **1005** (system-dependent initialization failed). A message is written to standard output indicating that system-dependent initialization failed and that details are in *CTSTATUS.FCS*.

New Features for Handling File Descriptor Limits

Several new features improve the ability of the system to handle situations concerning file descriptors:

- *Improved file descriptor server startup messages* (page 148)
- *Fail server startup if file descriptor limit can't be increased to required value* (page 149)
- *Write message to standard output when file descriptor limit is too low* (page 149)
- *New file descriptor limit compatibility keyword* (page 149)

Improved File Descriptor Limit Messages Logged During Server Startup

The file descriptor limit messages that FairCom Server logs to *CTSTATUS.FCS* at startup have been improved. The following information is logged:

- the file descriptor limit was successfully increased
- a warning that the maximum set by the system is not high enough to meet the server's file descriptor requirements
- the limit and the file descriptor requirement values

Now we also include the user limit in the file descriptor limit, because a TCP/IP socket uses a file descriptor.

Note: If the file descriptor limit cannot be increased to the required value, server startup fails. See *Fail server startup if file descriptor limit can't be increased to required value* (page 149).

Sample messages

Case #1: The system file descriptor limit is not high enough:

```
Mon Apr 28 13:17:55 2014
 - User# 00001  ERROR: The hard limit on file descriptors available to this process (4096) is lower than the database engine's file descriptor requirement (11275). Either increase the hard limit, or decrease the FILES or CONNECTIONS settings.
```

Case #2: c-treeACE was able to increase the limit:

```
Mon Apr 28 13:19:02 2014
 - User# 00001  Successfully increased current file descriptor limit to: 11275
```

Server Now Fails to Start if File Descriptor Limit Can't be Increased to Required Value

Appropriate operating system file descriptors are critical to c-treeACE server operation.

Note: If a file descriptor limit set by the operating system cannot be increased to the required value, server startup fails with error **1005** (system-dependent initialization failed).

If the file descriptor limit set by the operating system isn't high enough, it is possible to fail to open a transaction start file when performing a checkpoint, causing the server to terminate abnormally. This behavior avoids a runtime error by catching the insufficient file descriptor limit at server startup.

Message Written to Standard Output When File Descriptor Limit is too Low

When the file descriptor limit for FairCom Server (set by the operating system) is set too low, a message is written to standard output indicating that system-dependent initialization failed and that details are in *CTSTATUS.FCS*.

New file descriptor limit compatibility keyword

Although running FairCom Server with insufficient file descriptors can lead to errors opening files, we have added a compatibility keyword, `COMPATIBILITY FILE_DESCRIPTOR_LIMIT`, that restores the previous behavior in case it is not convenient for a system administrator to set the file descriptor limit for the FairCom Server process to the required value or it is not desired to decrease the FILES or CONNECTIONS settings.

Note: Use of this keyword is generally discouraged. It is provided for backward compatibility or short-term use until site administrators are able to increase file appropriate file descriptor limits for a FairCom Server process.

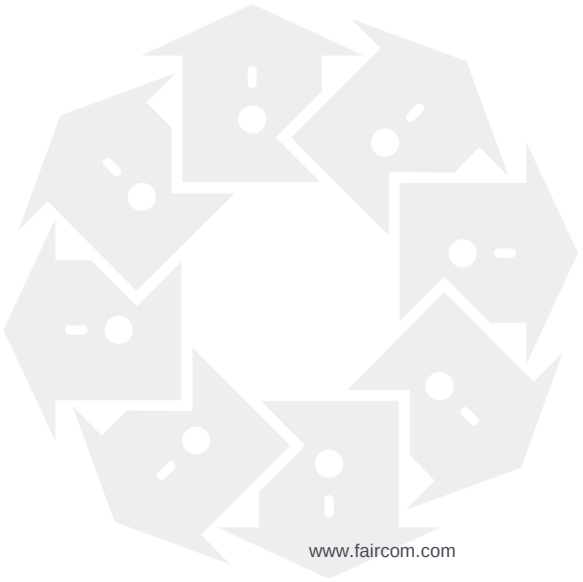
A message is also logged to *CTSTATUS.FCS* explaining that the `COMPATIBILITY FILE_DESCRIPTOR_LIMIT` configuration option can be used to allow the server to start in this situation:

```
Tue Apr 29 12:23:44 2014
 - User# 00001  ERROR: The hard limit on file descriptors available to this process (500) is lower than the database engine's file descriptor requirement (1043). Either increase the hard limit, or decrease the FILES or CONNECTIONS settings.
Tue Apr 29 12:23:44 2014
 - User# 00001  Note: The configuration option COMPATIBILITY FILE_DESCRIPTOR_LIMIT can be used to allow c-tree Server to start even if the file descriptor limit is insufficient. However, this can lead to errors opening files.
```

15. Advanced Data Integrity Controls

FairCom takes data integrity seriously.

c-treeACE V11 demonstrates our commitment to maintaining the safety of your data.



15.1 Linux File System Performance and Safety

Review the FairCom website for important information about c-treeACE performance and integrity on Linux platforms: **Linux Caching Considerations** (https://docs.faircom.com/doc**/ctserver/linux-caching-considerations.htm**).

15.2 Flush KEEPOPEN Files to Disk With Last File Close for Enhanced Data Integrity

For a data or index file that does not use full transaction logging (*ctTRNLOG*), FairCom Server now flushes updated cache pages and sets the update flag to zero for the file when the last user closes the file and the `KEEPOPEN_LIST` option keeps the file open.

The configuration option `KEEPOPEN_FLUSH NO` can be used in ***ctsrvr.cfg*** to disable this flushing behavior and revert to the behavior prior to V10.3.1, although this is not recommended.

This is a change from previous behavior: When FairCom Server's `KEEPOPEN_LIST` configuration option was in effect for a c-tree data or index file that was not under transaction control, FairCom Server kept that file open after the last user closed the file. Updated data and index cache pages remained and were not written to disk before the call to close the file returned to the caller. This caused unnecessary data integrity risk should the FairCom Server abnormally terminate while the file remained open. The file is likely to fail to open with an **FCRP_ERR** error, 14, under such scenarios.

15.3 LOKREC() modes to unlock all records in all files of the specified type that are open by the caller

LOKREC() function's *ctFREE_FILE* mode now supports options to free ALL locks in ALL specified types of files that a caller has open. To use this feature, call **LOKREC()** with mode of *ctFREE_FILE*, *recbyt* of zero (it is ignored), and set *datno* to one, or a combination of the following values:

- *ctFREEALL_NOTRAN* - free all non-transaction file locks
- *ctFREEALL_NOIICT* - free all transaction-controlled file locks without IICT enabled
- *ctFREEALL_IICT* - free all transaction-controlled file locks with IICT enabled
- *ctFREEALL_TRAN* - free all transaction-controlled file locks
- *ctFREEALL_ALL* - free all file locks

Example

Free all locks in all non-transaction files and all transaction-controlled files that are not using *IICT* that the caller has open.

```
rc = LOKREC( ctFREEALL_NOTRAN | ctFREEALL_NOIICT,ctFREE_FILE, 0 );
```

15.4 Prevent ISAM Index Key Value Updates

Beginning with V11, it is possible to create an index that does not allow an ISAM update to change the key values. On such an index, an ISAM record add can add a new key value and an ISAM record delete can delete a key value, but an ISAM record update cannot change a key value. (Note the special case for a variable-length record: if an ISAM update causes the record to move, the key is allowed to change its record offset from the old offset to the new offset; but the key value itself is not allowed to change.)

To create an index with this feature enabled, OR *KTYP_NOISMUPD* into the key type (*ikeytyp*) field of the *IIDX* structure for that index file before calling **CREIFIL()** to create the index.

PUTIFIL() can be used to turn this bit on or off. Changes take effect immediately.

An ISAM record update that attempts to change a key value for an index with this property enabled fails with error **UKEY_ERR** (1001).

15.5 New ctFeatKEEP_XFREED Lock Mode to Mark Entires in User Lock Table for Unlock Requests During a Transaction

This highly specialized lock mode is a great example of the precision engineering FairCom provides for ultimate data management and flexibility.

ctFeatKEEP_XFREED enables support for *ctMARK_XFREED* on transaction Begins and *ctKEEP_XFREED* on transaction Ends. *ctMARK_XFREED* causes unlock requests during a transaction to add a special attribute bit to the user lock table entry which indicates an explicit unlock request has been made. During a transaction End, *ctKEEP_XFREED* queries use this attribute bit to determine whether or not the lock should be released.

Note: *ctFeatKEEP_XFREED* is only available when TRANPROC is defined.

Default

ctFeatKEEP_XFREED is enabled by default in v10.5.0 and later.

15.6 Flush Directory Metadata to Disk for Transaction-Dependent File Creates, Deletes and Renames

When a file is created, renamed, or deleted, the new name of the file is reflected in the file system entry on disk only when the containing directory's metadata is flushed to disk. If the system crashes before the metadata is flushed to disk, the data for the file might exist on disk, but there is no guarantee that the file system contains an entry for the newly created, renamed, or deleted file. In a test case we noticed that after a system power loss a transaction log containing valid log entries still had the name of the transaction log template file.

In release V11 and later, c-tree ensures that creates, renames, and deletes of transaction log files and transaction-dependent files are followed by flushing of the containing directory's metadata to disk. This change also applies to other important files such as *CTSTATUS.FCS*, the master key password verification file, and files created during file compact operations (even if not transaction dependent).

To revert to the old behavior, add `COMPATIBILITY NO_FLUSH_DIR` to *ctsrvr.cfg*.

15.7 Permit ADMIN Group Member Access to Files with Corrupt Resource Chains

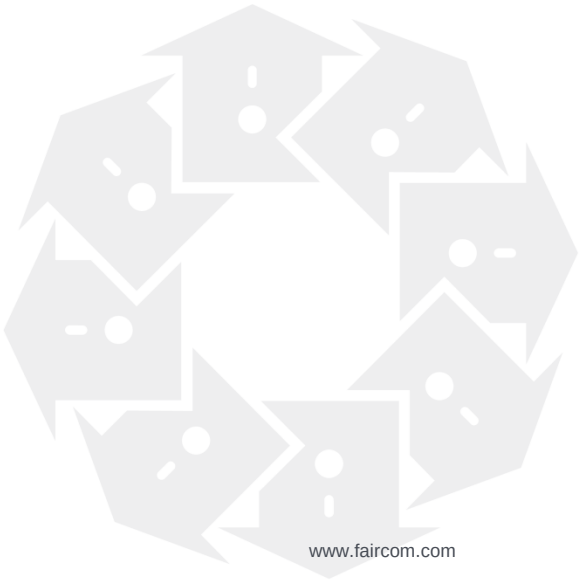
A file can now be opened even if it has an invalid resource chain. However, the *ctOPENCRPT* file mode is not sufficient to open a file in this state. Includes both *ctDISABLERES* and *ctOPENCRPT* to enable the file open to proceed.

On a server, the user opening the file must belong to the ADMIN group as passwords cannot be checked with resources disabled. The file mode will be forced to read-only (*ctREADFIL*).

Note: Some resources such as encryption and compression are essential to reading a file; if their resources cannot be processed then the file open operation will continue to fail.

16. Security Controls

FairCom continues to maintain a watchful eye on data security. We have long offered multiple data security controls, from user authentication controls to advanced data encryption. Authentication has been expended to include LDAP, Active Directory, and Native OS.



16.1 LDAP Authentication Controls and Group Support

c-treeACE now supports LDAP authentication for both ISAM and SQL user connections.

LDAP (Lightweight Directory Access Protocol) is a directory solution employed in many enterprise environments centralizing institutional data. That data includes user credentials, for example, allowing “single sign-on” to multiple systems, now including c-treeACE servers. This greatly simplifies administration of large numbers of users accessing many independent systems. And, c-treeACE allow support for both user and group management at SQL and NoSQL layers.

LDAP Group Membership Control

LDAP support includes an ability to check LDAP group membership. Specify the following configuration for the `LDAP_ISAM_ALLOWED_GROUP` and/or `LDAP_SQL_ALLOWED_GROUP` options using this syntax.

```
LDAP_ISAM_ALLOWED_GROUP {attr:ATTRIBUTE_VALUE}{base:BASE_VALUE}{filter:FILTER_VALUE}
```

For example:

```
LDAP_ISAM_ALLOWED_GROUP
{attr:member}{base:dc=mycompany,dc=com}{filter:(&(objectClass=groupOfNames)(cn=myusergroup))}
```

Important: The super administrator (ADMIN) user account is always authenticated using c tree's authentication and not LDAP authentication. This means a client not supporting LDAP/secure key exchange logon can still connect using the ADMIN account.

Compatibility Notes

- The c-treeACE implementation of LDAP support is based on the OpenLDAP API standard. Other LDAP implementations should function similarly based on the standard API calls used.
- c-treeACE SSL support included in securing LDAP connections is based on OpenSSL. Most standards-based SSL implementations should function similarly.
- C# and Java interfaces use support built in to their frameworks to implement secure key exchange. They do not require a separate SSL library.
- The C# interface requires *BigInteger* support, only available in .NET framework 4.0 and later.

16.2 Added Restrictions on Advanced Encryption Master Key Configuration Options

c-treeACE Server supports a master key file in which the advanced encryption master key can be stored. The `MASTER_KEY_FILE` configuration option is used to specify a local encrypted key store.

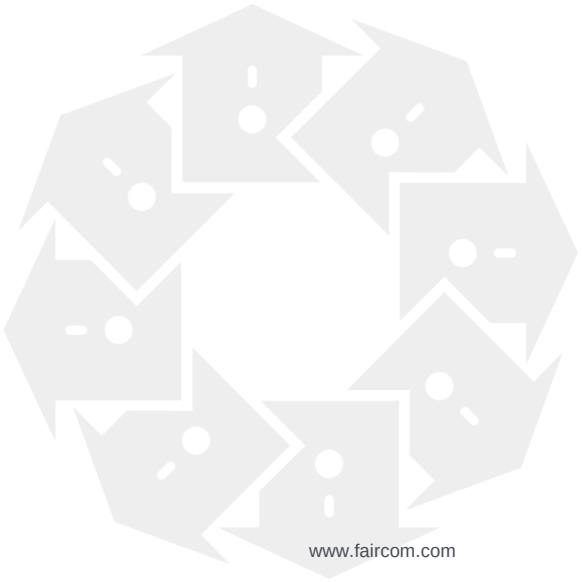
To prevent tampering with advanced encryption keys, the server configuration option `ALLOW_MASTER_KEY_CHANGE` must be specified to allow changing master passwords via the **SECURITY** API function (an administrative client-side API call). Default value: NO.

Note: The server configuration option `MASTER_KEY_FILE` now considers a value of NONE to indicate no master key file is in use. This option can be specified in a settings file to prevent a configuration file from using an existing master key file.

17. Interface Technology Additions

The interfaces are your application's way of accessing the many features and benefits of c-treeACE.

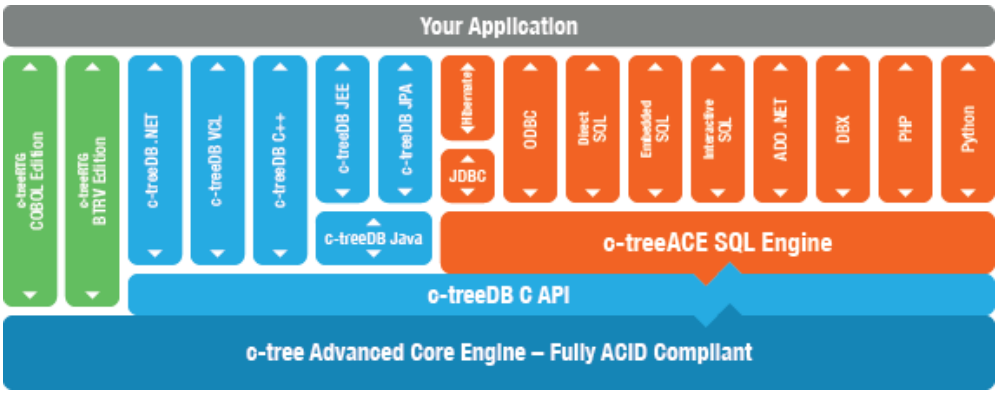
The FairCom APIs offer broad support for developers working in a variety of environments. This section lists changes in those APIs.



17.1 Overview of Current c-treeACE Interface Technology

c-treeACE continues to support the most current interface technologies available. Whatever your development platform, you'll find an interface solution readily available. Choose direct NoSQL ISAM APIs for control and performance. Use SQL APIs for connectivity and advanced SQL access. Over 20 total APIs available for the best fit with your application development environment.

And, of course, both are always simultaneously available over structured or unstructured data for ultimate No+SQL access!



17.2 c-treeACE NoSQL ISAM APIs

c-treeACE enjoys a broad range of API technologies for data access from many popular programming platforms and frameworks.

- c-treeDB C and C++
- c-treeDB.NET
- c-treeDB for Java
- C language ISAM API library

Choose the API best fitting for your development framework. And of course, you may always mix up your API access with the best match for your data needs.

FairCom.CtreeDb - Added ServerDateTime methods

.NET assemblies are available targeting multiple development strategies. FairCom DB API and ISAM frameworks are both supported.

New methods have been added to report server side date and time:

- The **ctdbServerDateTime(IntPtr Session)** method has been added to the `ctdb` class.
- The `CTDateTime` **GetServerDateTime()** method has been added to `CTSession` class

FairCom DB API for Java Now Supports MRT Tables

MRT (Multi-record tables) tables bring advanced relational access to tables with a varying schema over their records. Virtual Table support (MRT) has now been added to FairCom DB API for Java.

The following methods have been added to FairCom DB API Java:

```
static int AddMRTTable(CTHANDLE handle, String Name, String ParentName, int Info)
static int CreateMRTTable(CTHANDLE handle, String Name, String ParentName, int CreateMode, String Filter)
static int SetMRTTableDefaultIndex(CTHANDLE handle, int idxno, byte[] min, byte[] max, int method)
static int SetMRTTableFilter(CTHANDLE handle, String  Condition)
```

FairCom DB API - New key type for Deferred Index

In V11 and later, a new key type, *CTINDEX_DFRIDX*, has been added to indicate a deferred index.

Another new type, *CTINDEX_NOMOD*, was added to indicate an index with unmodifiable ISAM and FairCom DB API keys.

FairCom DB API Callback Updates for Types SDK

An array of function callbacks are available in FairCom DB API for table and field level control. These are extensively used by the c-treeACE SQL types SDK callback implementations. This is required, for example, when working with existing data types that are not compatible with expected SQL types. Several modifications were done to make this more effective across all data types.

When field callbacks are in place to manipulate field type and size, the SQL interface calls the following for LONG VARCHAR fields:

1. **ctdbGetFieldSize()** - Given a record handle and a field number, it returns its defined field size from the table definition.
2. **ctdbGetFieldDataLength()** - Given a record handle and a field number, it returns the field content size (the number of bytes actually being consumed by the data in the field).

To make these two functions work properly with field callback in place, these changes were made:

1. **ctdbGetFieldSize()** now takes advantage of the existing *CTDB_FIELD_GET_LENGTH* callback.
2. A new callback, *CTDB_FIELD_DATA_LEN*, is called by **ctdbGetFieldDataLength()**.

FairCom DB API Default Field Types Added

To ease default field definitions in FairCom DB API, a new API function with several default types have been added.

```
CTDBRET ctdbDECL ctdbSetFieldDefaultValueType(CTHANDLE Handle, CTDEF_TYPE def_type);
```

Where def_type is one of the following symbolic constants:

- `CTDEF_LITERAL`: the value is a literal
- `CTDEF_USER`: the current* user name
- `CTDEF_NULL`: null
- `CTDEF_SYSDATE`: the current* date
- `CTDEF_SYSTIME`: the current* time
- `CTDEF_SYSTIMESTAMP`: the current* timestamp

GetServerDateTime() Added to FairCom DB API for Java

A new function to get the current date and time, **ctdbServerDateTime()**, was implemented in the Java class CTSession as **CTSession::GetServerDateTime();**

Table Lock Mode for LOKREC

To acquire a file (table) lock using c-treeACE call **LOKREC()** (https://docs.faircom.com/doc**/ctreeplus/lockctdata.htm**) with a new *ctLOCK_FILE* mode as follows:

```
LOKREC(datno, ctLOCK_FILE, lockmode);
```

where:

- *datno* is the data file number
- *lockmode* is the desired locking mode, which can be any of the following values:

  ctREADREC - Acquire a file read lock. If the lock cannot be immediately acquired because a file write lock or record write locks exist on the file, then return an error.

  ctREADREC_BLK - Acquire a file read lock. If the lock cannot be immediately acquired because a file write lock or record write locks exist on the file, then block until those locks are released and the lock can be acquired.

  ctENABLE - Acquire a file write lock. If the lock cannot be immediately granted because file or record locks exist on the file, then return an error.

  ctENABLE_BLK - Acquire a file write lock. If the lock cannot be immediately acquired because file or record locks exist on the file, then block until those locks are released and the lock can be acquired.

To free a file lock, call **LOKREC()** as follows:

```
LOKREC(datno, ctUNLOCK_FILE, 0);
```

A file lock request can fail with the following errors:

- **FACS_ERR(26)** - The specified file number (*datno*) is not active.
- **FMOD_ERR(48)** - The specified file is a type of file that does not support file locks (for example, an index file, a superfile host, or a partitioned file).
- **DEAD_ERR(86)** - Could not block on file lock request due to deadlock.
- **TLOK_ERR(1025)** - Could not obtain table lock because the table is locked, or a request to lock the table is pending, or a conflicting data record read or write lock exists.

A record lock request that fails due to a table lock may encounter:

- **DLKT_COD(1024)** - Could not obtain data record lock because the table is locked, or a request to lock the table is pending (sysiocod).

A connection can only acquire one file lock on a particular file at a time.

A connection can acquire a table write lock when:

- no table write lock is held on the file

  and
- no table read lock is held on the file

  and
- either no read and write locks are held on the file, or only the table lock requester is holding read and/or write locks on the file.

A connection can acquire a table read lock when:

- no table write lock is held on the file

and

- no write locks are held on the file

When a table write lock request is granted, the requesters write and read locks are released. All lock waiters are made to wait for the table to be unlocked.

Record lock and unlock requests for a connection that holds a table write lock have no effect.

A connection that has acquired a table read lock can promote the table read lock to a table write lock by requesting a table write lock. The request is granted if:

- only one table read lock is held on the file
  and
- no read locks are held on the file
  and
- no record read or write lock or table lock requests are waiting.

Notes

- Blocked table lock requests take priority over record lock requests. If releasing a lock makes it possible for a blocked table lock to be acquired, any record lock requests that are waiting on the lock that is being released queue up waiting for the table lock to be released.
- Blocked table write lock requests take priority over blocked table read lock requests.
- When a table read lock request is granted, the requester's read locks are released. (Note that the requester has no write locks on the table when the table read lock request is granted; otherwise, the table read lock request would have been denied.) Record read lock requests for a connection that holds a table read lock have no effect. Record write lock requests for a connection that holds a table read lock are denied.
- A table lock is not supported on a superfile host, only on its data file members. A table lock request on a superfile host fails with error **FMOD_ERR(48)**.
- A table lock is not currently supported on a partition host or partition member file. A table lock request on these types of files fails with error **FMOD_ERR(48)**.

Table lock behavior with transactions

If a table write lock is in effect when a record is updated in a transaction, the table write lock cannot be removed until the transaction commits or aborts. Note that the call to free the table write lock returns success (NO_ERROR) and *sysiocod* is set to **UDLK_TRN (-3)** in this situation, indicating that the table write lock was not released.

A table lock cannot be released even if the transaction is restored to a savepoint that precedes the first update of a record in the locked table.

The *ctMARK_XFREED* and *ctKEEP_XFREED* transaction modes, used in calls to start and commit a transaction respectively, cause the commit to keep only the locks that were explicitly freed during the transaction. When these modes are used and a table lock is acquired before the transaction commits, the mode applies to the table lock: if a call is made to free the table lock before the transaction commits, the commit releases the table lock; otherwise, the commit keeps the table lock.

This feature is only available with the c-treeACE Server operational model.

New Xtd8 File Mode to Automatically Create Directories

A call to a c-treeACE *Xtd8* file create function can set the *ctAUTOMKDIR* bit in the *splval* field of the *xcreblk* parameter to enable directories that do not exist to be created when creating that file. Remember that a call to an *Xtd8* call with a non-NULL *xcreblk* requires that *xcreblk* points to an array of *XCREblk* structures, one for each physical file in the *IFIL* structure. The *splval* *ctAUTOMKDIR* bit affects only the corresponding file, and so if you want all the physical files that are being created in your *Xtd8* call to have their directories automatically created, you must set the *ctAUTOMKDIR* bit on the *splval* field for each *xcreblk* array entry that corresponds to a physical file.

See Also

- Automatic Directory Creation for New Files (page 132)

ctCopyFile

Description:

Server-to-server file copy.

ctCopyFile() makes a copy of a c-tree data or index file, or a c-tree data file and its associated index files. The caller specifies the source and destination file names and options that affect the behavior of the file copy. c-tree copies the files synchronously. That is, the copy call does not return until the file copy is completed.

Prototype:

```
NINT  ctCopyFile(pCPYFIL pcpyfil);
```

Parameters:

ctCopyFile() accepts one parameter: a pointer to a file copy option structure, CPYFIL (see definitions below).

Returns:

ctCopyFile() returns zero indicating all files were successfully copied or a non-zero value to indicate one or more files failed to copy. The caller can supply an optional error message buffer and buffer size. If an error message buffer is supplied and an error occurs, the file copy function writes an error message to the error message buffer.

Notes:

The caller can optionally specify a callback function that is periodically called for displaying user feedback progress bars. The callback function is called at the start of the file copy, for each 1% of the file that has been copied, and at the end of the file copy. The callback function must return zero to permit the file copy to continue, and it can return a non-zero error code in order to terminate the file copy operation.

To copy a file, a caller must be authorized by c-tree (including the file system) to open a file for read access. ADMIN user is always granted read permission for all files. (c-tree file permissions are only supported in the client/server model and not with standalone usage.)

c-tree attempts to open the file in exclusive mode, while still allowing other connections to open the file for read access. If the file is already open, c-tree flushes updates to disk and sets the update flag to zero to indicate that the file is in a clean, consistent state. If c-tree cannot open the file in exclusive mode, the copy operation fails.

An option provides a way to open the file in shared mode. This option indicates the server should try to open the files to copy in shared mode if the files can't be opened in exclusive mode. To use this option, OR the *ctCFallowSharedOpen* bit into the *cpoptions* field of the CPYPRM structure passed to **ctCopyFile()**

Note: Files with an *.FCS* extension (such as *FAIRCOM.FCS*, *SYSLOGDT.FCS*, *L0000001.FCS*, *SNAPSHOT.FCS*, etc.) are considered to be a c-tree internal system housekeeping files. The file copy function does not allow copying from or to c-tree internal system files. An attempt to copy such a file fails with error code 1013 (**SCPY_ERR**).

Copying c-tree files

When copying a c-tree file, c-tree assigns a new file ID to the copy of the file so that the file ID does not match the original file's ID, and c-tree sets the data and index file and directory names in the data file's *IFIL* resource to the names of the new copies of the files. Other than those differences, the copied file is identical to the original file: it contains the same active and deleted data records or index nodes and key values, resources, and has the same properties as the original file, including:

- security (file owner, file group, file permission mask, file password)
- data encryption
- data compression
- transaction control

The caller indicates what action to take when a file already exists with the same name as the specified destination file. By default, the file copy fails in this situation, but the caller can specify that the existing file should be deleted instead.

When copying multiple files (for example when copying a data file and its associated index files), if an error occurs c-tree stops the file copy operation and deletes all files that were successfully copied. If desired, the caller can specify an option to keep all the successfully copied files in case of an error.

Limitations

The following operations are not currently supported:

- Copying a partition host file.
- Copying a segmented file.
- Copying a superfile host file.
- Copying a non-c-tree file.

File Copy Options

ISAM or Low-Level Copy

Choose either an ISAM-level or a low-level file copy:

- An ISAM copy specifies the name of a c-tree data file, and the data file and its associated index files are copied.
- A low-level file copy copies only the specified file.

Use one of these options to select the desired behavior:

- *ctCFisam* - Copy an ISAM data file and its index files.
- *ctCFlowlevel* - Copy only the specified file.

Copy with file opened in shared mode

- *ctCFallowSharedOpen* - Allow files to be opened in shared mode if possible. Available in V11.6 and later.

Copy to Directory or File Names

Choose to either copy the file(s) to a specified directory or to copy the specified file(s) to the specified new file name(s).

Copying to a specified directory requires the caller to specify the source file name and the destination directory name. In order to use this option for an ISAM file copy, the data and index files must all reside in the same directory.

Copying to the specified new file names requires the caller to specify the source file name and the destination file name. When using this option for an ISAM file copy, the caller must supply source and destination file names for the data file and all its index files.

Use one of these options to select the desired behavior:

- *ctCFtoDir* - Copy files to the specified directory.
- *ctCFtoName* - Copy files to the specified file names.

Destination File Exists

Choose the desired behavior when the destination file exists. By default, if the destination file exists, the file copy function leaves the destination file unchanged and returns an error. The caller can instead choose to delete the destination file by using the following option:

- *ctCFifexistDelete* - Delete destination file if it exists.

Create Destination Directory

By default, if a destination directory does not exist, the file copy fails. The caller can choose to create the destination directory by using the following option:

- *ctCFcreateDirs* - Create destination directories.

Keep Successfully-Copied Files if One Fails

By default, when a c-tree data file and its associated index files are being copied, if one of the files fails to be copied, all the successfully-copied files are deleted. The caller can choose to keep the successfully-copied files by using the following option value:

- *ctCFiferrorKeep* - Keep all successfully-copied files if a subsequent file fails to be copied.

Remote File Copy

By default, a c-treeACE server creates the destination file on its own filesystem (a local file copy). Files can be copied to another server (a remote file copy) by specifying the following option and specifying the user ID, user password, and server name in the server parameter block or the appropriate API function to set server parameters:

- *ctCFtoServer* - Copy files to a specified server.

Your remote server must have `ENABLE_TRANSFER_FILE_API YES` in the configuration file or you will receive error **NSUP_ERR** (454, Feature not supported).

You will also need a multithreaded client library in the local server (*mtclient.dll* for Windows or *libmtclient.so* for Unix/Linux) or you will receive error **TR_CLIL_ERR** (850, "Transactional replication: Failed to start c-tree remote client subsystem: see *CTSTATUS.FCS* for details").

If you are logged on as a non-ADMIN user, you also need `COMPATIBILITY NONADMIN_TRANSFER_FILE` specified or you receive error **LADM_ERR** (589, Admin logon required).

Note: SYSCFG() has been updated with state bits to check for library availability as well as if the copy feature is enabled. Refer to **SYSCFG()** documentation for usage.

Verify

If you wish to verify that the source and destination files are identical after the file has been successfully copied, use the following option:

- *ctCFverify* - After copying a file, verify that the source and destination files are identical.

  Note: This option is not yet supported for a remote file copy operation.

File Copy Function Call Structure Definitions

```
typedef struct cpyfil {
    LONG     versn;     /* Structure version              */
    LONG     ncpyprm;   /* Number of file copy param lists */
    LONG     errbufsiz; /* Size of optional error buffer   */
    LONG     reserved;  /* Reserved for future use         */
    pCPYPRM  pcpyprm;   /* File copy parameter lists       */
    pCPYFNC  pcpyfnc;   /* Callback function               */
    pTEXT    errbuf;    /* Optional error message buffer   */
} CPYFIL, ctMEM * pCPYFIL;
```

`versn` must be initialized to value CPYFIL_VERS_CUR.

File Name Structure Definition:

```
typedef struct ctfnam {
    TEXT     name[MAXFNAME]; /* Directory or file name.    */
} CTFNAM, *pCTFNAM;
```

File Copy Parameter Structure Definition:

```
typedef struct ctcpyprm {
    LONG     versn;     /* Structure version              */
    ULONG    cpoptions; /* File copy options              */
    ULONG    fboptions; /* File block options             */
    LONG     filnamcnt; /* Number of file names           */
    pTEXT    fileword;  /* File password                  */
    SRVPRM   srvprm;    /* Server parameters              */
    pCTFNAM  srcfilnam; /* Source file names              */
    pCTFNAM  dstfilnam; /* Destination dir or file names   */
} CPYPRM, ctMEM * pCPYPRM, ctMEM ** ppCPYPRM;
```

`versn` must be initialized to value CPYPRM_VERS_CUR

File Copy Server Parameter Definition:

```
typedef struct srvprm {
    NINT     versn;          /* Structure version   */
    TEXT     userid[32];     /* User id             */
    TEXT     userword[64];   /* User password       */
    TEXT     servname[256];  /* Server name         */
} SRVPRM, ctMEM * pSRVPRM;
```

`versn` must be initialized to value SRVPRM_VERS_CUR.

File Copy Callback Function Parameters:

```
typedef struct cpystt {
    pCPYFIL pcpyfil; /* options passed to file copy function */
    pTEXT   filnam;  /* name of the source file being copied */
    COUNT   val;     /* percent complete or error code       */
    TEXT    opcode;  /* operation code                       */
} CPYSTT, *pCPYSTT;
```

File Copy Callback Function Prototype:

```
typedef NINT (*pCPYFNC)(pCPYSTT pcpystt);
```

File Copy Callback Function Operation Codes:

- *ctCPYCBKbeginfile* - Start file copy
- *ctCPYCBKprogfile* - File copy progress (val contains the percent complete)
- *ctCPYCBKendfile* - End file copy (val contains the error code)
- *ctCPYCBKvrfyfile* - File verify progress (val contains the percent complete)

Notes:

1. If there is only one physical index file associated with a data file and the physical index file's *aidxnam* is NULL (so that it has the same name as the data file), the *ctCFtoName* option can be used with only one set of names (old data file name and new data file name).
2. When a file copy fails because the destination file exists, error **18** is returned if a destination index file exists and error **19** is returned if the destination data file exists.

Error Codes

New error codes introduced with the **ctCopyFile()** function:

Value	Symbolic Constant	Explanation
1013	**SCPY_ERR**	The file copy API function does not allow copying a FairCom system file.
1014	**DCPY_ERR**	When using the file copy API option to copy files to a new directory, the source data and index files must all reside in one directory.
1015	**CBKTRM_ERR**	A user-defined callback function terminated the operation.
1016	**GFSZ_ERR**	Error getting file size. Check *sysiocod* for system error code value.
1017	**FDIF_ERR**	The file comparison detected differences between the two files.

Existing error codes that the **ctCopyFile()** function can return:

Value	Symbolic Constant	Explanation
43	**FVER_ERR**	The version of the CPYFIL, CPYPRM, or SRVPRM structure passed to the file copy function is not supported.

See Also

- *Copy Files Between c-treeACE Servers* (page 144, */doc/ctreeplus/63314.htm*)
- COMPATIBILITY COPY_FILE_OPEN_SHARED (https://docs.faircom.com/doc/ctserver/compatibility-copy-file-open-shared-config.htm)
- **ctfcpAllocateHandle** (page 186)
- **ctfcpFreeHandle** (page 184)
- **ctfcpAddFileCopyOperation** (page 185)
- **ctfcpRemoveFileCopyOperation** (page 187)
- **ctfcpSetErrorBuffer** (page 188)
- **ctfcpGetErrorBuffer** (page 189)
- **ctfcpSetCallback**
- **ctfcpSetCopyOptions** (page 191)
- **ctfcpSetCopyFilenames** (page 192)
- **ctfcpSetCopyFilePassword** (page 193)
- **ctfcpSetServerParameters** (page 194)
- **ctfcpCopyFile** (page 195)

ctRecordUpdateCallbackControl

Declaration

ctRecordUpdateCallbackControl() is used to add and delete callback function definitions. Its prototype is as follows:

```
NINT  ctDECL ctRecordUpdateCallbackControl(pRUCBCTL prucbctl);
```

Description

A file can have one or more callback function definitions. Each callback function definition is identified by its name, which is a case-sensitive ASCII string. The callback functions are called in the order in which they were added.

These callbacks execute within the server process space. Programming errors will affect the entire database server process.

The synchronous callback types (*RUCBonrecupd*, *RUCBontrancmt*) occur in the context of the calling user. Most c-tree APIs cannot be safely called from these callbacks, as they may alter existing internal user state information. For advanced usage, information could be sent to another thread to make c-tree file calls if desired. ctThrd* APIs are useful in this regard to pass information to another thread.

The asynchronous callback types (*RUCBqueuethrd*, *RUCBqueueapp*) are more flexible, since they don't share the same state as the original operation. It should be safe to directly call c-tree file APIs, however, ensure *you don't operate on the same file that is generating the callback*.

More generally, within the server each thread has an implicit top level "connection" context that is single threaded, which we call OWNER. If you want to do something like have a pool of worker threads, you'll need to manage this using **ctSetOWNER** (faircom.com) or **ctdbGetCtreeOWNER()**/**ctdbSetCtreeOWNER()**. Only a single thread should be within a c-tree API call with a particular OWNER at the same time.

The *RUCBCTL* structure has the following definition:

```
typedef struct rucbctl {
    COUNT   version;    /* version of this structure */
    COUNT   opcode;    /* operation code  */
    LONG    bufsiz;    /* buffer size   */
    pVOID   bufptr;    /* input/output buffer  */
#ifndef ct8P
    LONG    pad;
#endif
} RUCBCTL, *pRUCBCTL;
```

The *RUCBACB* structure, used when adding a callback definition, has the following definition:

```
/* Input buffer format for RUCBCTLaddcallback opcode. */
typedef struct rucbacb {
    COUNT   version;    /* Version of this structure       */
    FILNO   datno;     /* Data file number        */
    LONG    calltm;    /* Time of function call       */
    LONG    nfncnames; /* Number of function names       */
    LONG    pad;       /* Padding to ensure 8-byte alignment of next field */
    pTEXT   datnam;    /* Data file name        */
    pTEXT   cbname;    /* Unique name to identify this callback     */
    pTEXT   dllname;   /* Name of callback DLL        */
    pTEXT   params;    /* Optional parameters        */
    ppTEXT  fncnames;  /* Callback function names       */
#ifndef ct8P
 LONG pad2[5];
#endif
} RUCBACB, *pRUCBACB;
```

The *calltm* member defines "when" a callback is executed. There are currently four possible values:

- *RUCBonrecupd* - Synchronous call during record update.
- *RUCBontrancmt* - Synchronous call during transaction commit.
- *RUCBqueuethrd* - Asynchronous queue update to a background thread that will call the callback after the transaction commits.
- *RUCBqueueapp* - Asynchronous queue update to the transaction log or memory queue. The application is responsible for reading the log/queue.

The *RUCBDCB* structure, used when deleting a callback definition, has the following definition:

```
/* Input buffer format for RUCBCTLdelcallback opcode. */
typedef struct rucbdcb {
    COUNT   version;    /* Version of this structure                   */
    FILNO   datno;     /* Data file number                           */
    LONG    pad;       /* Padding for 8-byte pointer alignment        */
    pTEXT   datnam;    /* Data file name                              */
    pTEXT   cbname;    /* Unique name to identify this callback       */
#ifndef ct8P
    LONG    pad2[2];
#endif
} RUCBDCB, *pRUCBDCB;
```

Option to Update All Existing Records

The default when adding a callback is to not call it for existing records. An option can be specified to choose a new behavior of calling the callback for all existing records. This option, *RUCBcallexist*, is specified by OR'ing it into the *calltm* field of the RUCBACB structure when calling **ctRecordUpdateCallbackControl()**.

Additional Callback for pre-RUC Updates

V13 and later supports an additional callback that is called at the start of record add and update operations and allows the callback to update the record image. By comparison, the record update callbacks that existed prior to this revision are called later in the add and update routines and do not support the callback changing the record image.

To use this feature, when calling **ctRecordUpdateCallbackControl()** to add a record update callback function, set the *calltm* field of the RUCBACB structure to *RUCBprerecupd*.

Recall that the prototype for the user-defined record update callback function is as follows:

```
NINT rucbRecordUpdateCallbackFunction(pRUCBF prucbf, pRUCBO prucbo, pRUCBSTT prucbstt);
```

Additional fields were added to the RUCBSTT structure. When the structure version (indicated by the *verson* field value) is RUCBSTT_VERSION_V02, the RUCBSTT structure includes fields that hold the record length, record image pointer, and function pointers to the server's memory allocation and free functions:

```
/* State information to pass to record update callback function: */
typedef struct rucbstt_t {
    LONG verson; /* structure version */
    VRLEN reclen; /* record length in bytes */
    LONG8 tranno; /* transaction number for the operation */
    pTEXT recbuf; /* buffer containing record image */
    rucbAllocFunction_t allocfn; /* memory allocation function */
    rucbFreeFunction_t freefn; /* memory free function */
} RUCBSTT, *pRUCBSTT;
```

To update the record image in your user-defined record update callback function, if you are increasing the record length, use the memory allocation function to reallocate the record buffer and set *prucbstt->recbuf* to the new buffer and *prucbstt->reclen* to the new record length. If you are not increasing the record length, you can modify the record image in the existing record buffer.

Note that for a fixed length data file or a *ctAugmentedFxd* variable length data file, the record update callback function is not allowed to change the record length. Attempting to do so causes the add or update operation to fail with error **DSIZ_ERR** (443).

Also, record update callbacks now receive the partition host file name and file number. The record update callback functions were receiving the file name and file number of the partition member. We have changed the behavior so that the host name and file number are passed if available. In our testing, we found that the host info might not be available when closing a partition member because the host might already be closed.

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

Follow these steps to use **ctRecordUpdateCallbackControl()** to add a callback function:

1. Declare *RUCBCTL* and *RUCBACB* structures and a list of callback names:

```
RUCBCTL rucbctl;
RUCBACB rucbacb;
pTEXT   myCallbackFunctionNames[NbrRUCBFnc] = {
    "rucbOpenFileCallback",
    "rucbCloseFileCallback",
    "rucbRecordUpdateCallback"
};
NINT    rc;
```

2. Initialize the *RUCBCTL* structure to indicate that this is an add callback operation:

```
memset(&rucbctl,0,sizeof(rucbctl));
rucbctl.verson = RUCBCTL_VERS_V01;
rucbctl.opcode = RUCBCTLaddcallback;
rucbctl.bufsiz = sizeof(rucbacb);
rucbctl.bufptr = &rucbacb;
```

3. Initialize the *RUCBACB* structure with your callback settings:

```
memset(&rucbacb,0,sizeof(rucbacb));
rucbacb.verson = RUCBACB_VERS_V01;
rucbacb.opcode = RUCBCTLaddcallback;
/* datno of -1 instructs ctRecordUpdateCallbackControl() to open the
file;
** otherwise it is the file number of an already-open data file. */
rucbacb.datno = -1;
rucbacb.datnam = "mydatafile.dat";
rucbacb.dllname = "ctuser.dll";
rucbacb.params = "my parameters";
rucbacb.fncnames = myCallbackFunctionNames;
rucbacb.nfncnames = NbrRUCBFnc;
rucbacb.cbname = "My First Callback";
/* This record update callback will be called on record update. */
rucbacb.calltm = RUCBonrecupd;
```

4. Call **ctRecordUpdateCallbackControl()** and check the result:

```
if ((rc = ctRecordUpdateCallbackControl(&rucbctl)) != NO_ERROR)
    printf("Error: Failed to add record update callback function '%s':
%d (%d)\n",
        rucbacb.cbname,rc,sysiocod);
else
```

```
    printf("Successfully added record update callback function
'%s'.\n",
        rucbacb.cbname);
```

To use **ctRecordUpdateCallbackControl()** to delete a callback function:

1. Declare *RUCBCTL* and istructures:
```
RUCBCTL rucbctl;
RUCBDCB rucbdcb;
NINT    rc;
```

2. Initialize the *RUCBCTL* structure to indicate that this is a delete callback operation:
```
memset(&rucbctl,0,sizeof(rucbctl));
rucbctl.verson = RUCBCTL_VERS_V01;
rucbctl.opcode = RUCBCTLdelcallback;
rucbctl.bufsiz = sizeof(rucbdcb);
rucbctl.bufptr = &rucbdcb;
```

3. Initialize the *RUCBDCB* structure with your callback settings:
```
memset(&rucbdcb,0,sizeof(rucbdcb));
rucbdcb.verson = RUCBDCB_VERS_V01;
/* datno of -1 instructs ctRecordUpdateCallbackControl() to open the
file;
** otherwise it is the file number of an already-open data file. */
rucbdcb.datno = -1;
rucbdcb.datnam = "mydatafile.dat";
rucbdcb.cbname = "My First Callback";
```

4. Call **ctRecordUpdateCallbackControl()** and check the result:
```
if ((rc = ctRecordUpdateCallbackControl(&rucbctl)) != NO_ERROR)
    printf("Error: Failed to delete record update callback function
'%s': %d (%d)\n",
        rucbdcb.cbname,rc,sysiocod);
else
    printf("Successfully deleted record update callback function
'%s'.\n",
        rucbdcb.cbname);
```

ctDeferredIndexControl

Declaration

Function prototype:

```
NINT ctDeferredIndexControl(pDFKCTL pdfkctl);
```

Description

The **ctDeferredIndexControl()** function can be used for monitoring and controlling deferred indexing.

The function accepts one parameter, a pointer to a deferred key control structure:

```
typedef struct dfkctl {
    COUNT   version; /* version of this structure */
    COUNT   opcode; /* operation code            */
    LONG    bufsiz; /* buffer size               */
    pTEXT   bufptr; /* output buffer             */
#ifndef ct8P
    LONG    pad;
#endif
} DFKCTL, *pDFKCTL;
```

To call this function set version to *DFKCTL_VERS_CUR* to use the current structure version from your c-tree header files, or specify a specific structure version such as *DFKCTL_VERS_V01* (version 1 of this structure, which is currently the only version supported).

Set *opcode* to one of the following values:

- *DFKCTLclearstats* - Clear deferred indexing statistics
- *DFKCTLgetstats* - Get deferred indexing statistics
- *DFKCTLgetstate* - Get deferred indexing thread state
- *DFKCTLgetbilstate* - Get background index load thread state
- *DFKCTLpausethrds* - Pause deferred indexing threads
- *DFKCTLresumethrds* - Resume deferred indexing threads

For opcodes *DFKCTLgetstats* and *DFKCTLgetstate*, set *bufptr* to point to a *DFKSTT* structure (defined below), and set *bufsiz* to the size of this buffer.

Statistics and thread state variables are returned in a structure of the following format:

```
/* deferred indexing statistics */
#define DFKSTTvern 1 /* DFKSTT (deferred indexing stats) version # */
typedef struct dfkstt {
    ULONG   client_ver;  /* client version of structure         */
    ULONG   server_ver;  /* server version of structure         */
    LONG8   tstamp;      /* Time stamp: seconds since 70        */
    LONG8   opnok;       /* Successful file opens               */
    LONG8   addok;       /* Successful adds                      */
    LONG8   delok;       /* Successful deletes                  */
    LONG8   rwtok;       /* Successful updates                  */
    LONG8   opnerr;      /* Failed file opens                   */
    LONG8   adderr;      /* Failed adds                         */
    LONG8   delerr;      /* Failed deletes                      */
    LONG8   rwterr;      /* Failed updates                      */
    LONG8   addskp;      /* Skipped adds                        */
    LONG8   delskp;      /* Skipped deletes                     */
    LONG8   rwtskp;      /* Skipped updates                     */
    LONG    qcnt;        /* Queue entries                       */
    LONG    lognum;      /* Log number of current scan pos      */
    LONG    logpos;      /* Offset of current scan pos          */
    DFRKOP  curopT;      /* Current operation for tran thread   */
    DFRKOP  curopN;      /* Current operation for non-tran thread*/
} DFKSTT, *pDFKSTT;
```

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

1) Read deferred indexing statistics:

```
NINT    rc;
DFKCTL  dfkctl;

memset(&dfkctl,0,sizeof(dfkctl));
dfkctl.verson = DFKCTL_VERS_V01;
dfkctl.opcode = DFKCTLgetstats;
dfkctl.bufsiz = sizeof(dfkstt);
dfkctl.bufptr = (pTEXT) &dfkstt;
if ((rc = ctDeferredIndexControl(&dfkctl)) != NO_ERROR) {
    printf("Error: Failed to get deferred indexing statistics: %d\n", rc);
} else {
    /* display dfkstt structure members */
}
```

2) Read deferred indexing thread state:

```
NINT    rc;
DFKCTL  dfkctl;

memset(&dfkctl,0,sizeof(dfkctl));
dfkctl.verson = DFKCTL_VERS_V01;
dfkctl.opcode = DFKCTLgetstate;
dfkctl.bufsiz = sizeof(dfkstt);
dfkctl.bufptr = (pTEXT) &dfkstt;
if ((rc = ctDeferredIndexControl(&dfkctl)) != NO_ERROR) {
    printf("Error: Failed to get deferred indexing statistics: %d\n", rc);
} else {
    /* display dfkstt structure members */
}
```

3) Clear deferred indexing statistics:

```
NINT    rc;
DFKCTL  dfkctl;

dfkctl.verson = DFKCTL_VERS_V01;
dfkctl.opcode = DFKCTLclearstats;
if ((rc = ctDeferredIndexControl(&dfkctl))) {
    printf("Error: Failed to reset deferred indexing statistics: %d\n", rc);
} else {
    printf("Successfully reset deferred indexing statistics.\n");
}
```

4) Pause deferred indexing threads.

```
NINT    rc;
DFKCTL  dfkctl;

dfkctl.verson = DFKCTL_VERS_V01;
dfkctl.opcode = DFKCTLpausethrds;
if ((rc = ctDeferredIndexControl(&dfkctl))) {
    printf("Error: Failed to pause deferred indexing threads: %d\n", rc);
} else {
    printf("Successfully paused deferred indexing threads.\n");
}
```

5) Resume deferred indexing threads.

```
NINT    rc;
DFKCTL  dfkctl;

dfkctl.verson = DFKCTL_VERS_V01;
dfkctl.opcode = DFKCTLresumethrds;
if ((rc = ctDeferredIndexControl(&dfkctl))) {
    printf("Error: Failed to resume deferred indexing threads: %d\n", rc);
} else {
    printf("Successfully resumed deferred indexing threads.\n");
}
```

ctThrdSharedCritical API for Scalable Read Locks

This release extends the threading API for reader/writer lock support that allow uncontended read locks for excellent speed and scalability. Shared lock performance should scale linearly. Exclusive lock performance is expected to be slower than alternatives.

NINT ctThrdSharedCriticalInit(ctSCRIT * scrit)

Initializes the shared critical pointed to by *scrit*.

Returns 0 on success.

NINT ctThrdSharedCriticalEnter(ctSCRIT * scrit, volatile RESERVATION * ticket)

A shared lock of *scrit*. Each calling thread must have a unique RESERVATION they provide to each *scrit*. There are multiple restrictions on *ticket*:

1. The RESERVATION must initialize as shown prior to first calling this function:
   `*ticket = CRIT_UNRESERVED`
2. Behavior is undefined if the RESERVATION is modified at any time after first use and prior to calling **ctThrdSharedCriticalCancelReservation**
3. Behavior is undefined if ticket becomes invalid memory prior to calling **ctThrdSharedCriticalCancelReservation** for this *scrit* and *ticket*.
4. Behavior is undefined if *ticket* is used by a different thread.
5. Behavior is undefined if the thread exits without calling *ctThrdSharedCriticalCancelReservation* for this *scrit* and *ticket*.

Behavior is undefined if called recursively.

Waits indefinitely if an Exclusive lock is in effect.

Note: False sharing of RESERVATION memory could impact scalability.

Returns 0 on success.

NINT ctThrdSharedCriticalExit(ctSCRIT * scrit, volatile RESERVATION * ticket)

Releases the shared lock of *scrit*. The same *ticket* passed by this thread to **ctThrdSharedCriticalEnter** must be used.

Returns 0 on success.

NINT ctThrdSharedCriticalCancelReservation(ctSCRIT * scrit, volatile RESERVATION * ticket)

Releases resources in *scrit* associated with *ticket*. This must be called prior to thread exit once a particular ticket has been passed to **ctThrdSharedCriticalEnter**.

Returns 0 on success.

NINT ctThrdSharedCriticalEnterExcl(ctSCRIT * scrit)

An exclusive lock of *scrit*. Behavior is undefined if called recursively.

Returns 0 on success.

NINT ctThrdSharedCriticalExitExcl(ctSCRIT * scrit)

Releases an exclusive lock of *scrit*.

Returns 0 on success.

NINT ctThrdSharedCriticalTerm(ctSCRIT * scrit)

Releases resources used by shared critical. Behavior is undefined if *scrit* is still in use by other threads.

ctVerifyFile

This ISAM-level function verifies the total integrity of a c-treeACE data file and its associated index files.

Function Name

```
NINT ctVerifyFile(pctVFYFIL pvfyfil);
```

Type

ISAM-level function

Description

- *pvfyfil* points to a *ctVFYFIL* structure (see ***ctport.h***)

```
typedef struct ctvfyfil {
    LONG    verson;          /* Version of this structure.            */
    LONG    options;         /* Which operations to perform.          */
    LONG    errbufsiz;       /* Size of optional error message buffer. */
    LONG    chkkey;          /* Index files to check (0=all, 1=first
                                index in IFIL, 2=second index, etc).   */
    pTEXT   datnam;          /* Name of data file to check.           */
    pTEXT   fileword;        /* Optional data file password.          */
    pVFYMSGCBFNC fnmessage;  /* Optional message callback function.   */
    pVFYPRGCBFNC fnprogress; /* Optional progress callback function.  */
    pVFYERRCBFNC fnerror;    /* Optional progress callback function.  */
    pTEXT    errbuf;         /* Optional error message buffer.        */
} ctVFYFIL, *pctVFYFIL;
```

An optional callback can be called from the *fnprogress* structure member.

Supported options

ctVFopenExclusive	Open the files in exclusive mode. Otherwise, the files are opened in read-only mode, allowing other connections to also open the file in read-only mode.
ctVFchkDataOnly	Read the data file in physical order, not checking keys.
ctVFchkDataWithKeys	Read the data file in physical order and check that each index contains the key built from the record image.
ctVFchkIndexInternals	Perform more thorough index validations for unusual problems.
ctVFchkIndexOnly	Read the index file in key order and don't check the key built from the record image.
ctVFchkIndexWithRecord	Read the index file in key order and check that the key built from the record image matches the key in the index.

ctVFchkDeletedSpace	Scan a data files delete stack or space management index verifying each deleted space entry corresponds to a deleted (not active) record.
ctVFYFILspaceStart	Deleted space scan is starting. (for use with a callback)
ctVFYFILspacePct	Deleted space scan in progress, with the *pct* parameter indicating approximate percent complete. (for use with a callback)
ctVFYFILspaceEnd	Deleted space scan has completed. (for use with a callback)

Note: **ctVerifyFile()** requires exclusive access to the file (*ctVFopenExclusive*). The only exception is the *ctVFchkDeletedSpace* option, which does not require exclusive access.

The source code for the accompanying **ctvfyfil** utility provides a good example showing how to use the **ctVerifyFile()** function.

Note: When a user-defined **ctVerifyFile()** error callback function returns a non-zero error code so that it can cancel the verification operation, **ctVerifyFile()** returns the c-tree error code and system error code for the error that triggered that call to the error callback function.

See Also

- **ctvfyfil Utility** (https://docs.faircom.com/doc**/ctreeplus/ctvfyfil-util.htm**)
- **ctVERIFYidx()** (https://docs.faircom.com/doc**/ctreeplus/ctverifyidx.htm**)

CloseConnection API Function to Cleanly Shut Down a Forked Connection

Forking a process with an active server connection causes the child and parent processes to each have a reference to the same connection. A new function, **CLCONN()**, was introduced to clean up one copy of this connection without breaking the other connection.

Declaration

```
ctCONV  COUNT ctDECL CLCONN( VOID );
```

Short name

CLCONN()

Description

When a process with an active c-tree connection is forked and does not call **exec()**, **CLCONN** should be called by one of the resulting processes to clean up an existing connection rather than calling **CLISAM()**.

This function is only available in client models.

ctfcpFreeHandle

Description:

Frees the specified file copy handle and all resources that it references.

Prototype:

```
NINT ctfcpFreeHandle(FCPHND fcphnd);
```

Parameters:

- *fcphnd* [IN] - The file copy handle to be freed.

Returns:

- NO_ERROR Success.
- **PNUL_ERR** - The specified file copy handle is NULL.

See also

- ctCopyFile (https://docs.faircom.com/doc/ctreeplus/ctcopyfile.htm)
- Copy Files Between c-treeACE Servers (page 144, /doc/ctreeplus/63314.htm)

ctfcpAddFileCopyOperation

Description:

Adds a file copy operation to the specified file copy handle.

Prototype:

```
NINT ctfcpAddFileCopyOperation(FCPHND fcphnd,pFCPOPR pfcpopr);
```

Parameters:

- *fcphnd* [IN] - The file copy handle to be freed.
- *pfcpopr* [OUT] - Pointer to memory in which the file copy operation handle is returned.

Returns:

- NO_ERROR - Success.
- **PNUL_ERR** - The specified file copy handle or file copy operation handle output pointer is NULL.
- **UALC_ERR** - Unable to allocate memory for the file copy operation.

See also

- ctCopyFile (https://docs.faircom.com/doc/ctreeplus/ctcopyfile.htm)
- Copy Files Between c-treeACE Servers (page 144, /doc/ctreeplus/63314.htm)

ctfcpAllocateHandle

Description:

Allocates a file copy handle.

Prototype:

```
NINT ctfcpAllocateHandle(pFCPHND pfcphnd);
```

Parameters:

- *pfcphnd* [OUT] - Pointer to memory in which the file copy handle is returned.

Returns:

- NO_ERROR - Success.
- **PNUL_ERR** - The output pointer for the file copy handle is NULL.
- **UALC_ERR** - Unable to allocate memory for the file copy handle.

See also

- ctCopyFile (https://docs.faircom.com/doc/ctreeplus/ctcopyfile.htm)
- Copy Files Between c-treeACE Servers (page 144, /doc/ctreeplus/63314.htm)

ctfcpRemoveFileCopyOperation

Description:

Removes a file copy operation from the specified file copy handle.

Prototype:

```
NINT ctfcpRemoveFileCopyOperation(FCPOPR fcpopr);
```

Parameters:

- *fcpopr* [IN] - The handle of the file copy operation to be removed.

Returns:

- NO_ERROR - Success.
- **PNUL_ERR** - The specified file copy operation handle is NULL.

See also

- ctCopyFile (https://docs.faircom.com/doc/ctreeplus/ctcopyfile.htm)
- Copy Files Between c-treeACE Servers (page 144, /doc/ctreeplus/63314.htm)

ctfcpSetErrorBuffer

Description:

Set the error buffer for a file copy handle.

Prototype:

```
NINT ctfcpSetErrorBuffer(FCPHND fcphnd,pTEXT errbuf,LONG errbufsiz);
```

Parameters:

- *fcphnd* [IN] - The file copy handle.
- *errbuf* [IN] - The error buffer.
- *errbufsiz* [IN] - The size in bytes of the error buffer.

Returns:

- NO_ERROR - Success.
- **PNUL_ERR** - The specified file copy handle is NULL.
- **NLEN_ERR** - The specified error buffer size is negative.

See also

- ctCopyFile (https://docs.faircom.com/doc/ctreeplus/ctcopyfile.htm)
- Copy Files Between c-treeACE Servers (page 144, /doc/ctreeplus/63314.htm)

ctfcpGetErrorBuffer

Description:

Gets the error buffer for a file copy handle.

Prototype:

```
pTEXT ctfcpGetErrorBuffer(FCPHND fcphnd,pNINT perrcod);
```

Parameters:

- *fcphnd* [IN] The file copy handle.
- *perrcod* - Memory address at which the error code is stored if an error occurs. Can be NULL.

Returns:

- Non-NULL value - Success
- NULL value - Either the error buffer is NULL, or an error occurred. If an error occurred, **perrcod* holds the error code.

See also

- ctCopyFile (https://docs.faircom.com/doc/ctreeplus/ctcopyfile.htm)
- Copy Files Between c-treeACE Servers (page 144, /doc/ctreeplus/63314.htm)

ctfcpSetCallback

Description:

Sets a callback function to be called to report progress as files are being copied.

Prototype:

```
NINT ctfcpSetCallback (FCPHND fcphnd,pCPYFNC pcpyfnc)
```

Parameters:

- *fcphnd* [IN] - The file copy handle.
- *pcpyfnc* - The file copy callback function.

Returns:

- NO_ERROR - Success.
- **PNUL_ERR** - The specified file copy handle is NULL.

See also

- ctCopyFile (https://docs.faircom.com/doc/ctreeplus/ctcopyfile.htm)
- Copy Files Between c-treeACE Servers (page 144, /doc/ctreeplus/63314.htm)

ctfcpSetCopyOptions

Description:

Set the file copy options for a file copy operation.

Prototype:

```
NINT ctDECL ctfcpSetCopyOptions(FCPOPR fcpopr,LONG options);
```

Parameters:

- *fcpopr* [IN] - The file copy operation handle.
- *options* [IN] - The file copy options.

Returns:

- NO_ERROR - Success.
- **PNUL_ERR** - The specified file copy operation handle is NULL.

See also

- ctCopyFile (https://docs.faircom.com/doc/ctreeplus/ctcopyfile.htm)
- Copy Files Between c-treeACE Servers (page 144, /doc/ctreeplus/63314.htm)

ctfcpSetCopyFilenames

Description:

Sets the source and destination filenames for a file copy operation.

Prototype:

```
NINT ctfcpSetCopyFilenames(FCPOPR fcpopr,NINT totnames,pCTFNAM srcnames,pCTFNAM dstnames);
```

Parameters:

- *fcpopr* [IN] - The file copy operation handle.
- *totnames* [IN] - The number of names.
- *srcnames* [IN] - Array of pointers to <totnames> source file names.
- *dstnames* [IN] - Array of pointers to <totnames> destination file names.

Returns:

- NO_ERROR - Success.
- **PNUL_ERR** - The specified file copy operation handle, source filename pointer, or destination filename pointer is NULL.
- **UALC_ERR** - Unable to allocate memory for the source and destination filename buffers.

See also

- ctCopyFile (https://docs.faircom.com/doc/ctreeplus/ctcopyfile.htm)
- Copy Files Between c-treeACE Servers (page 144, /doc/ctreeplus/63314.htm)

ctfcpSetCopyFilePassword

Description:

Sets the password to be used when opening the file that is being copied.

Prototype:

```
NINT ctfcpSetCopyFilePassword(FCPOPR fcpopr,pTEXT fileword);
```

Parameters:

- *fcpopr* [IN] - The file copy operation handle.
- *fileword* [IN] - The file password.

Returns:

- NO_ERROR - Success.
- **PNUL_ERR** - The specified file copy operation handle is NULL.

See also

- ctCopyFile (https://docs.faircom.com/doc/ctreeplus/ctcopyfile.htm)
- Copy Files Between c-treeACE Servers (page 144, /doc/ctreeplus/63314.htm)

ctfcpSetServerParameters

Description:

Sets the user name, password, and server name to use if copying files to another server.

Prototype:

```
NINT ctfcpSetServerParameters(FCPOPR fcpopr,pTEXT userid,pTEXT userword,pTEXT servername);
```

Parameters:

- *fcpopr* [IN] - The file copy operation handle.
- *userid* [IN] - The user ID.
- *userword* [IN] - The user's password.
- *servername* [IN] - The server name.

Returns:

- NO_ERROR - Success.
- **PNUL_ERR** - The specified file copy operation handle is NULL.

See also

- ctCopyFile (https://docs.faircom.com/doc/ctreeplus/ctcopyfile.htm)
- Copy Files Between c-treeACE Servers (page 144, /doc/ctreeplus/63314.htm)

ctfcpCopyFile

Description:

Copy files using the options that are set in the specified file copy handle.

Prototype:

```
NINT ctfcpCopyFile(FCPHND fcphnd);
```

Parameters:

- *fcphnd* [IN] - The file copy handle.

Returns:

- NO_ERROR - Success.
- **PNUL_ERR** - The specified file copy handle is NULL.
- **PBAD_ERR** - Invalid parameter specified. For example, the file copy parameter count is zero or negative.
- **UALC_ERR** - Unable to allocate memory for the source and destination filename buffers.
- Other errors - This function can return any error that the **ctCopyFile** API function returns.

17.3 c-treeACE SQL APIs

c-treeACE SQL boasts a wide range of interface technologies, including many of the most popular platforms.

- ODBC
- JDBC
- ADO.NET
- PHP
- Python
- C language Direct SQL API

And of course stored procedures, triggers and user defined functions can be written in Java and now .NET with c-treeACE SQL V11.

Database Management Methods Added to ADO.NET Data Provider

The following new database management methods were added to support Entity Framework 6:

```
public static void CreateDatabase(string connectionString)
public static void DropDatabase(string connectionString)
```

These new methods are optional although they are required by some Entity Framework tests. Additional fixes were also made to types returned by Entity Framework.

JDBC - Character Set Can Now Be Specified in the Connection URL

In V11 and later, special "high-bit" characters (ASCII > 127) can be properly displayed in c-treeACE SQL JDBC-based tools. The JDBC driver encodes all strings as UTF-16 because that is the format used by Java. Prior to this change, the source was assumed to be UTF-8. It is now possible to specify the character set in the URL so it can be handled properly.

The character set can be specified in the connection URL as follows:

```
jdbc:ctree:port@host_name:db_name?characterEncoding=encoding
```

The URL string has the following components:

- *jdbc:ctree* - An identifying protocol and subprotocol string for the c-treeACE SQL JDBC Driver.
- *:port* - The port number associated with the c-treeACE SQL server on the host system.
- *@host_name* - Name of the server system where the database resides.
- *:db_name* - Name of the database.
- *?characterEncoding=encoding* - Replace *encoding* with a valid Java encoding name (e.g., *US-ASCII*, *ISO-8859-1*, *UTF-8*, etc.).

c-treeACE SQL JDBC Now Allows Specifying User and Password in Connection URL

In V11 and later, the c-treeACE SQL JDBC driver allows specifying the user and the password in the connection URL using the following syntax:

```
jdbc:ctree:port@host_name:db_name?user=username&password=passwd
```

The URL string has the following components:

- *jdbc:ctree* - An identifying protocol and subprotocol string for the c-treeACE SQL JDBC driver.
- *:port* - The port number associated with the c-treeACE SQL server on the host system.
- *@host_name* - Name of the server system where the database resides.
- *:db_name* - Name of the database.
- *?settings* - Setting as per the following table separated by "`&`"

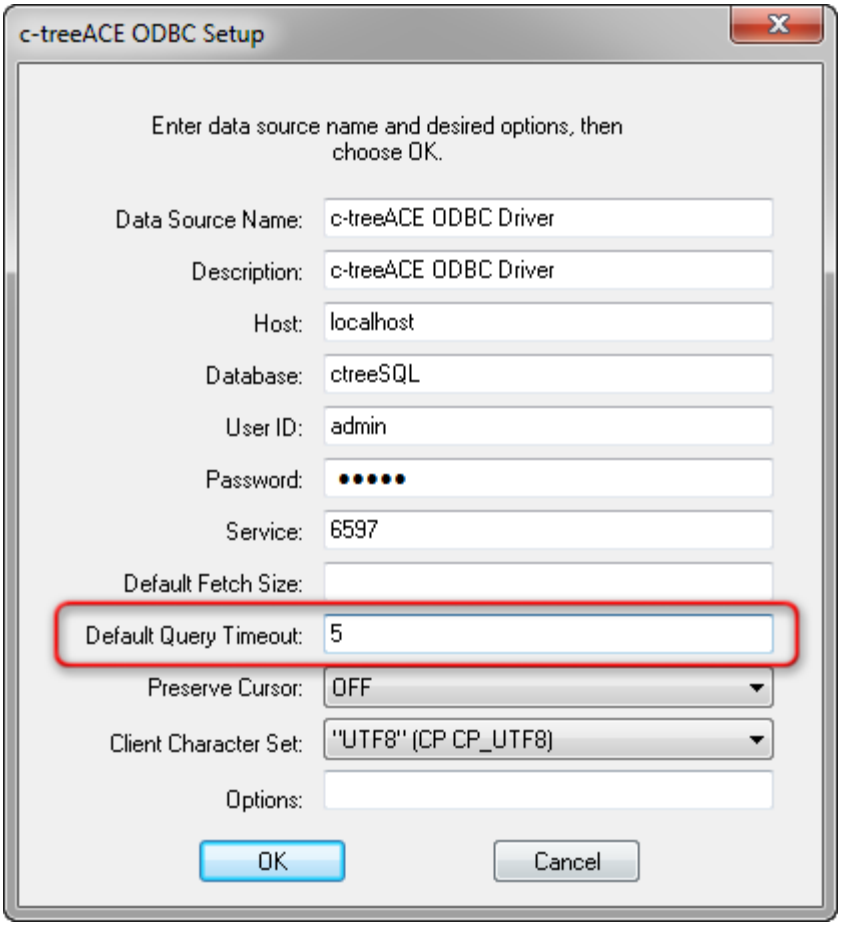
URL settings:

- *characterEncoding=encoding* - Encoding is a valid java encoding name.
- *user=username* - Where *username* is a valid database user.
- *password=passwd* - Where *passwd* is the user password.

The user and password specified in the URL take precedence over other ways of specifying them in JDBC.

c-treeACE SQL ODBC Ability to Set Query Timeout in DSN and Connection String

It is now possible to set the **Default Query Timeout** (in seconds) in the DSN and in the connection string. This value can be set when configuring the DSN on Windows by using the new **Default Query Timeout** field in the c-treeACE Setup dialog:



In the connection string, the attribute is:

- `QUERY_TIMEOUT=[number of seconds]`

In the *ODBC.INI* file, the attribute is:

- `Default Query Timeout=[number of seconds]`

Settings in the connection string take precedence over the setting in the DSN/*ODBC.INI*.

c-treeACE SQL ODBC Ability to Set "Default Fetch Size" in DSN or Connection String

It is possible to set a "Default Fetch Size" in bytes in the DSN. This value is the size used internally by the driver to fetch multiple rows from the server. This setting reduces network requests resulting in performance gains. If not set, the internal buffer size is 5000 bytes.

In your connection string, set the attribute `"FETCH_SIZE=[number of bytes]"`

In your *ODBC.INI* file, set the attribute `"Default Fetch Size=[number of bytes]"`

Connection string settings take precedence over DSN/ODBC.INI settings.

c-treeACE SQL ODBC Unix ODBC driver for AIX and Solaris

In V11 and later, c-treeACE SQL ODBC drivers for Unix are available on AIX and Solaris (Sparc) platforms.

c-treeACE SQL Direct SQL ctsqlGetParameterName()

Support was added in V11 to be able to retrieve the name of a parameter (which makes sense only for named parameters, i.e. `":param"`). The following function was added to the c-treeACE SQL Direct SQL API (*ctsqlapi.dll*) to retrieve parameter names:

Function:

```
ctsqlGetParameterName
```

Description:

Retrieve the parameter name (when using named parameter `":name"`).

Parameters:

- *hCmd* [IN] - Command handle
- *index* [IN] - Parameter number. This must be a number greater or equal to zero but less than the parameter count.

Return:

Returns the parameter name.

c-treeACE SQL Direct SQL ctsqlIsParameterNull

In V11 and later, DSQL exposes a method to verify if a parameter has been set to NULL.

Function:

```
ctsqlIsParameterNull
```

Retrieve the parameter.

Parameters:

- *hCmd* [IN] - Command handle.
- *index* [IN] - Parameter number. Must be a number greater or equal to zero but less than parameter count.

Return:

Return CTSQL_TRUE if the parameter is set to NULL.

c-treeACE SQL Direct SQL ctsqlGetNumericParameterAsString

In V11 and later, DSQL exposes a method to retrieve a numeric parameter as a string.

Function

```
ctsqlGetNumericParameterAsString
```

Retrieve the Numeric parameter as string.

Parameters

- *hCmd* [IN] - Command handle.
- *index* [IN] - Parameter number. Must be a number greater or equal to zero but less than parameter count.
- *buffer* [IN] - String buffer pointer.
- *length* [IN] - String buffer size.

Return

Return error code or 0 on success.

Example

```
CTSQLRET ctsqlDECL ctsqlGetNumericParameterAsString(pCTSQLCMD hCmd, INTEGER index, CTSQLCHAR* buffer,
INTEGER length)
```

c-treeACE SQL Direct SQL sqlda cursors

A new function, **ctsqlNewDACursor**, was added. This function provides a convenient way to access sqlda information through a high-level API without the need to use the c-treeACE SQL Direct SQL infrastructure.

The **ctsqlNewDACursor** function takes a *sqlda* as a parameter and returns a pCTSQLCURSOR. The *sqlda* passed in must stay valid until the cursor is freed. It is the programmer's responsibility to free the cursor and the *sqlda*.

Limitations on the CTSQLCURSOR:

- Operations on LVB/LVC/BLOB/CLOB columns return **SQL_ERR_FENOTSUP** error.
- **ctsqlNext** moves within the boundary of the *sqlda* (i.e., can be used to move to the next record in the sqlda); once the records in *sqlda* are finished it returns **CTSQL_NOTFOUND**.
- The *sqlda* is considered to contain only 1 record even if it contains multiple records.

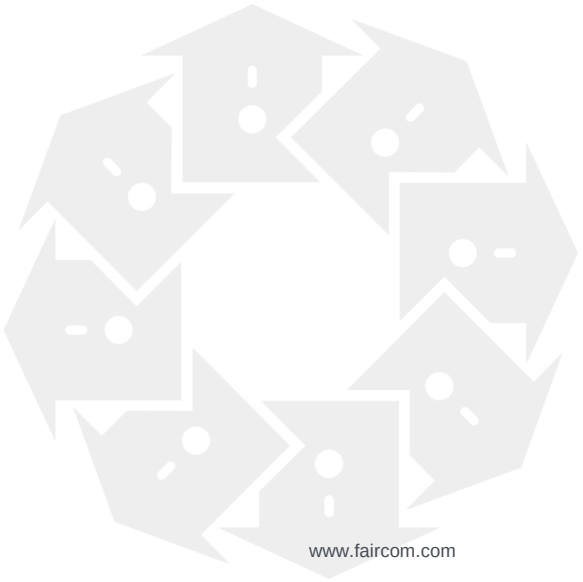
Note: These functions are intended for internal API usage and are documented solely for completeness of API available.

17.4 Embarcadero (Borland) XE - Support for 64-bit VCL Drivers

As part of our commitment to "no platform left behind," support has been added for 64-bit VCL drivers for Embarcadero (Borland) XE.

18. Latest News for GUI Tools

Graphical tools remain one of the most popular additions to c-treeACE. They get another refresh and continue to be available in cross-platform Java, our original .NET version for Windows, and now we add exciting new support for browser-based versions of these valuable tools.



18.1 Latest in Tools Development

We've addressed many of your suggestions and bug reports with hundreds of fixes. Update to V11 and take advantage of all the latest available features.

New Browser Based Tools!

SQL Explorer (page 204)

Replication Management Studio *(Contact FairCom for latest availability.)*

.NET Based Tool Updates

.NET framework (V2) is no longer supported by .NET versions of the GUI tools. Framework 3.5 remains supported.

Configuration files have been added allowing .NET tools to execute on machines with .NET Framework 4.x without requiring .NET Framework 3.5. Framework 4.x is supported under the following conditions:

1. If Framework 3.5 is also installed in the machine no .config files are needed.
2. If Framework 3.5 is NOT installed in the machine the .config files ARE needed.

Java Based Tools

Cross platform Java compatibility remains. Java based tools should remain compatible with Java 1.6 and later.

Scriptable Command-Line Tools

Many routine tasks are best accomplished with a scripted command-line utility (page 225). FairCom continues to advance and support these necessary tools. Let us know your requirements!

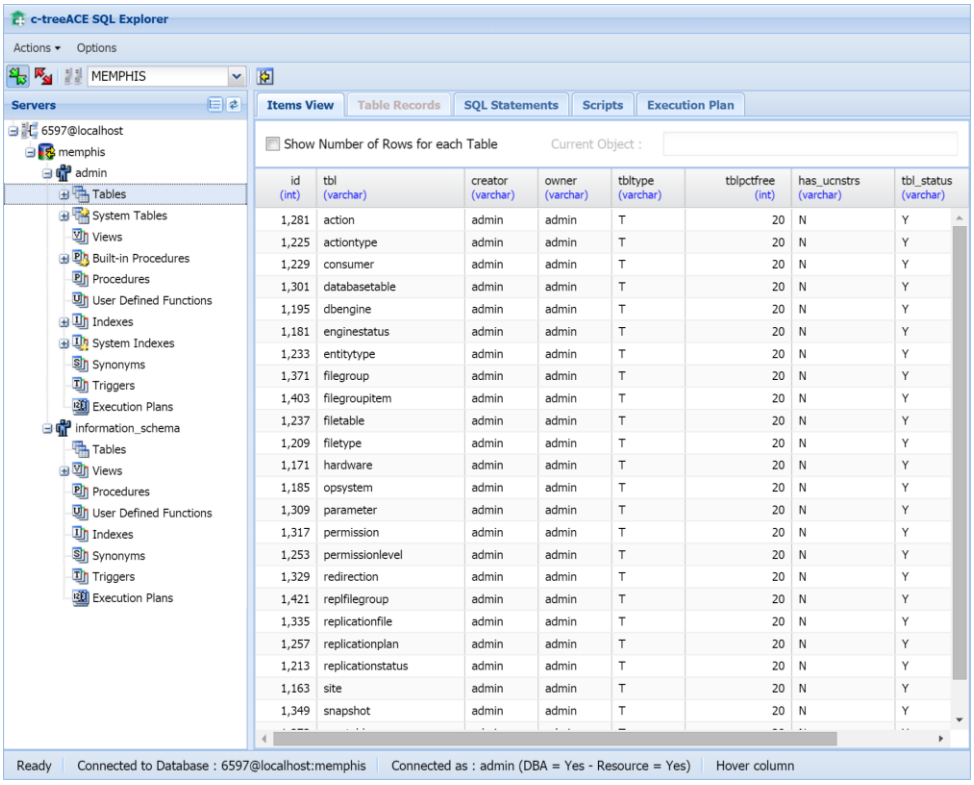
18.2 Browser Based c-treeACE SQL Explorer

c-treeACE now includes an embedded web server for remote browser administration.

All web components including Javascript files and *.css* templates are contained in a single *\tools* folder found under the working directory of your c-treeACE SQL installation. This is the default location the embedded web service looks for components. This is configurable as the document root configuration.

Simply point your browser to *http://<fully.qualified.domain.name>/tools* to access your c-treeACE web service. The embedded web service uses default web port 8080 (or 8443 for SSL). You'll be presented with a familiar logon prompt.

Contact FairCom for latest availability and updates!



It is also possible to use your existing Apache or Microsoft IIS web servers. Simply copy the *\tools* folder to your local web server and access from there. Install c-treeACE SQL PHP support and you can access your c-treeACE Server via your web server from any browser. *Be sure to change your administrator password for secure access!*

> *Currently, web based c-treeACE server access requires a working c-treeACE SQL PHP installation.*

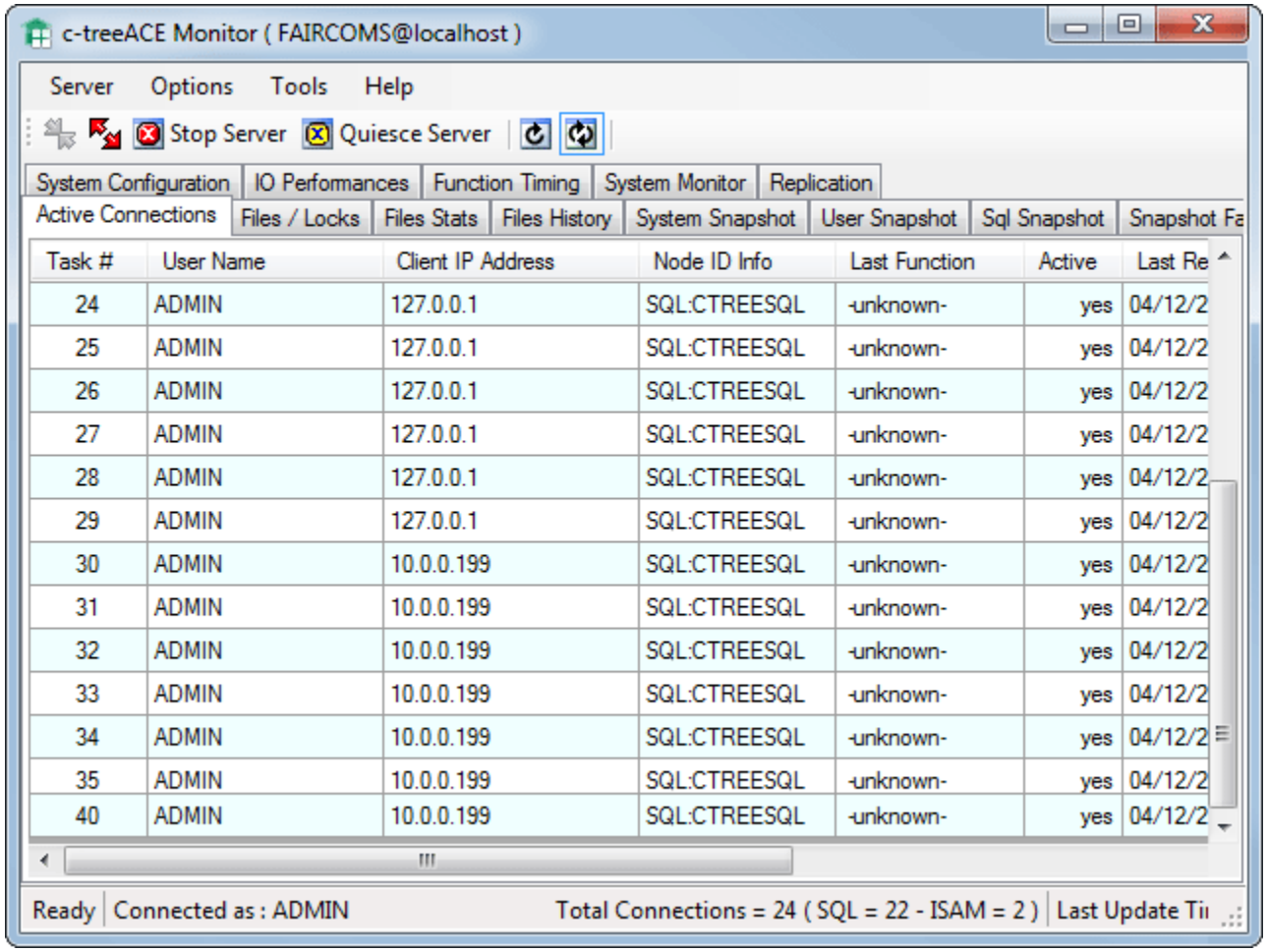
Look for more upcoming browser-based utilities in the future!

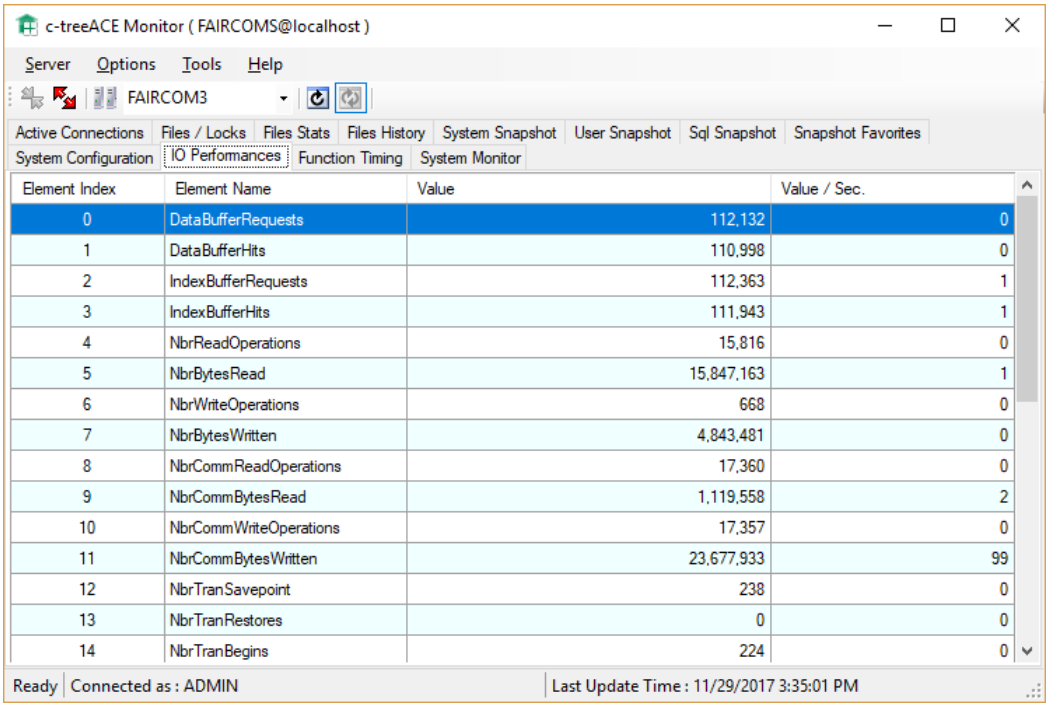
18.3 c-treeACE Monitor

c-treeACE Monitor is a comprehensive tool for observing the many hundreds of server metrics. View users and lock interactions, I/O metrics, c-tree API timings and much much more.

- **Collect Function Timing Statistics**
- **Advanced File Usage Statistics**
- **Quiesce c-treeACE**
- **View Current Configuration Options**

- **Disconnect a User**





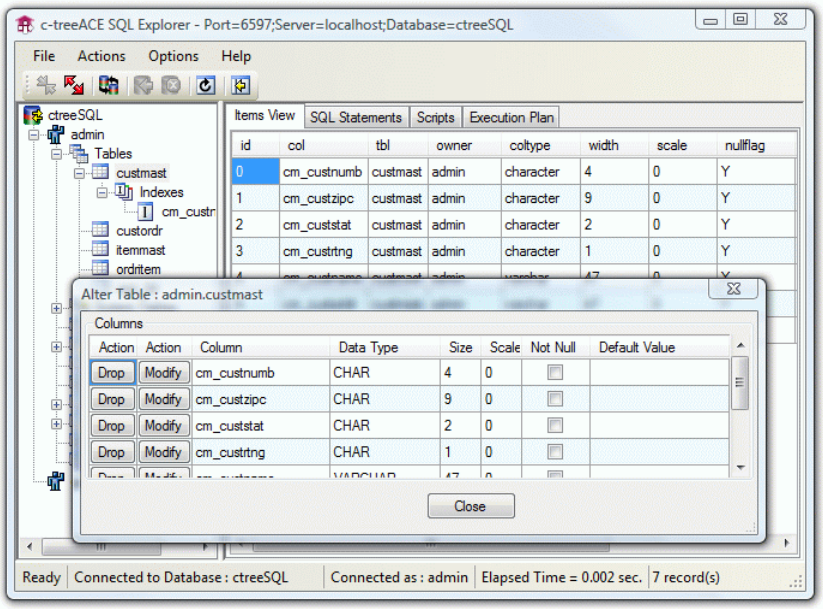
Updated in V11

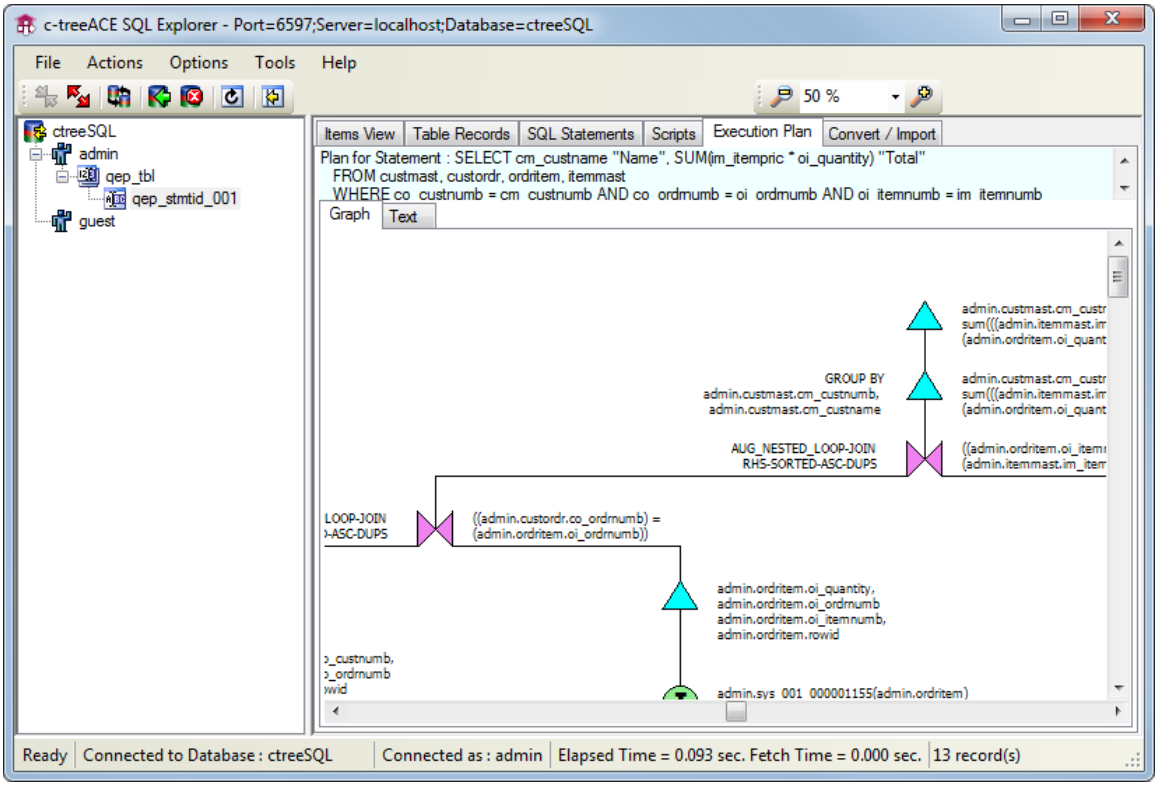
- A new **Node ID Info** column was added to the **Users for file** and **Locks for file** grids displaying the Node ID assigned for that connection.
- A new column has been added to the **Function Timing** grid showing time spent in a given interval by each function.
- The **Stop Server** button has been moved to avoid inadvertently shutting down a server It is now located closer to the **Quiesce Server** button.For further safety, the confirmation dialog now defaults to Cancel.
- FairCom RTG functions are now displayed in the Java and .NET versions in the **Function Timing** panel.

18.4 c-treeACE SQL Explorer

c-treeACE SQL Explorer is a comprehensive tool to view and manage all objects contained in your database. Create new databases, tables, indexes and constraints. Even view and manage your stored procedures, triggers and UDFs Export to comma separated values is also possible for use with other needs.

- **Execute Custom c-treeACE SQL Statements**
- **Load, Edit and Run c-treeACE SQL Scripts Interactively**
- **View query execution plans to profile and examine complex queries for optimization analysis**
- **Create and Manage Databases**
- **Create, Alter and Drop Tables, Views, and Indexes**
- **Create and Drop Stored Procedures and Triggers**
- **And Much More**





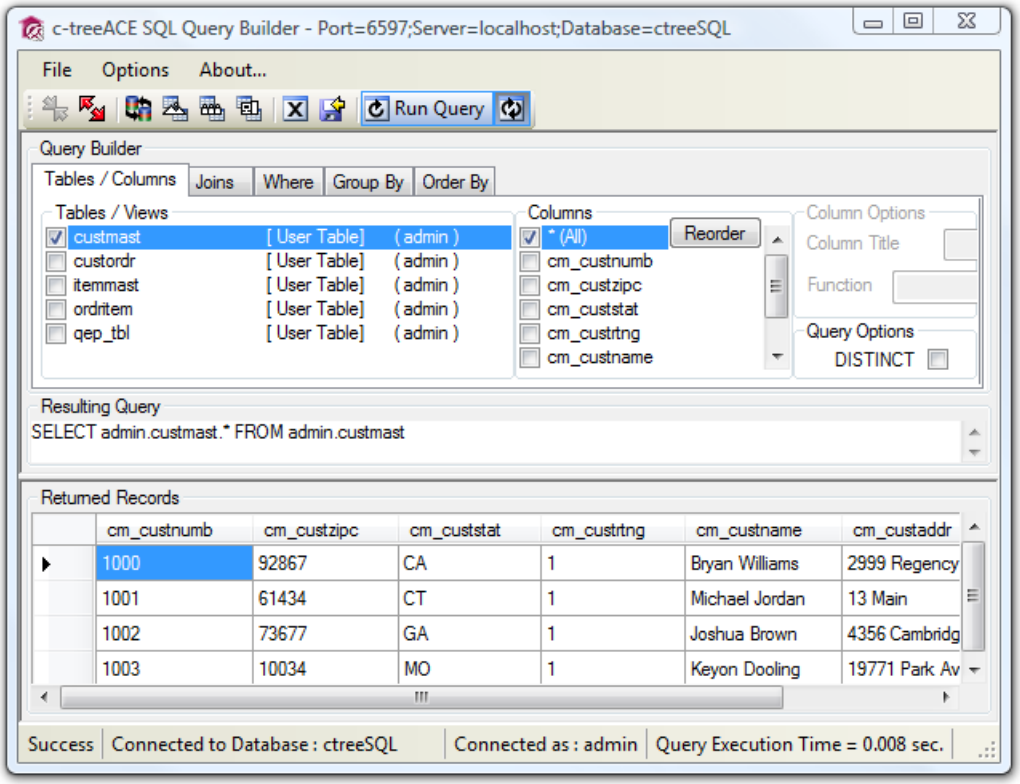
Updated in V11

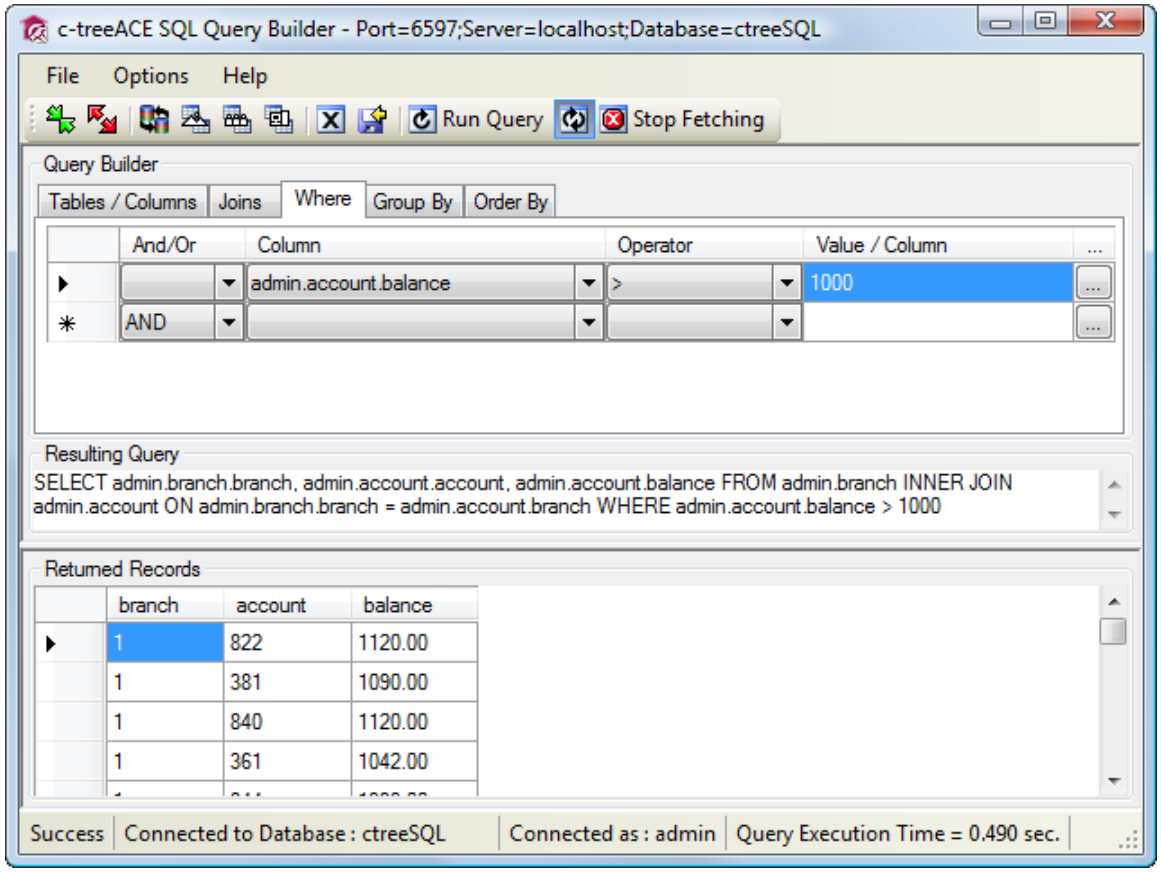
- An option was added to the Server Manager dialog in the .NET version of the c-treeACE SQL Explorer to allow setting a specific Charset for SQL connections.
- Greatly extended CSV data export options.
- Handling of comments inserted in the statement window has been greatly improved.
- CTRL+A now selects all the text within the SQL Statement window.
- Support for millisecond Time and Timestamp values has been added.

18.5 c-treeACE SQL Query Builder

c-treeACE SQL Query Builder makes it very easy to create sophisticated queries against your SQL data. Pick and choose your tables and columns, set your join conditions, and quickly view the resulting retrieved data.

- **Quickly Build and Test Complex Queries**
- **Dynamic Selection of Tables and Columns**
- **Easy Joins and with Drop Down Selections**

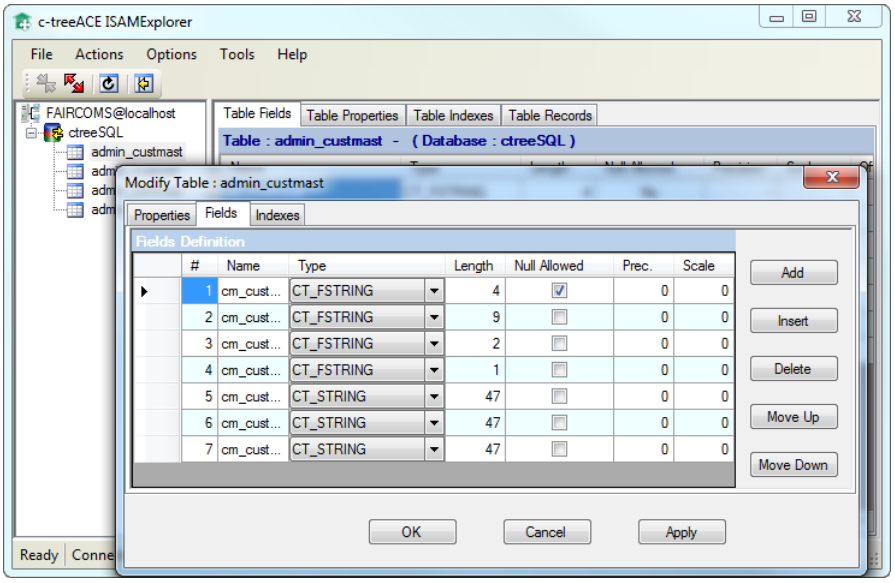




18.6 c-treeACE ISAM Explorer

The multi-purpose c-treeACE Explorer includes database views for ISAM tables.

- **Create and Drop Databases**
- **Create, Alter and Drop Tables**
- **Create and Drop Indexes**
- **Add Existing Files**
- **Browse Data**



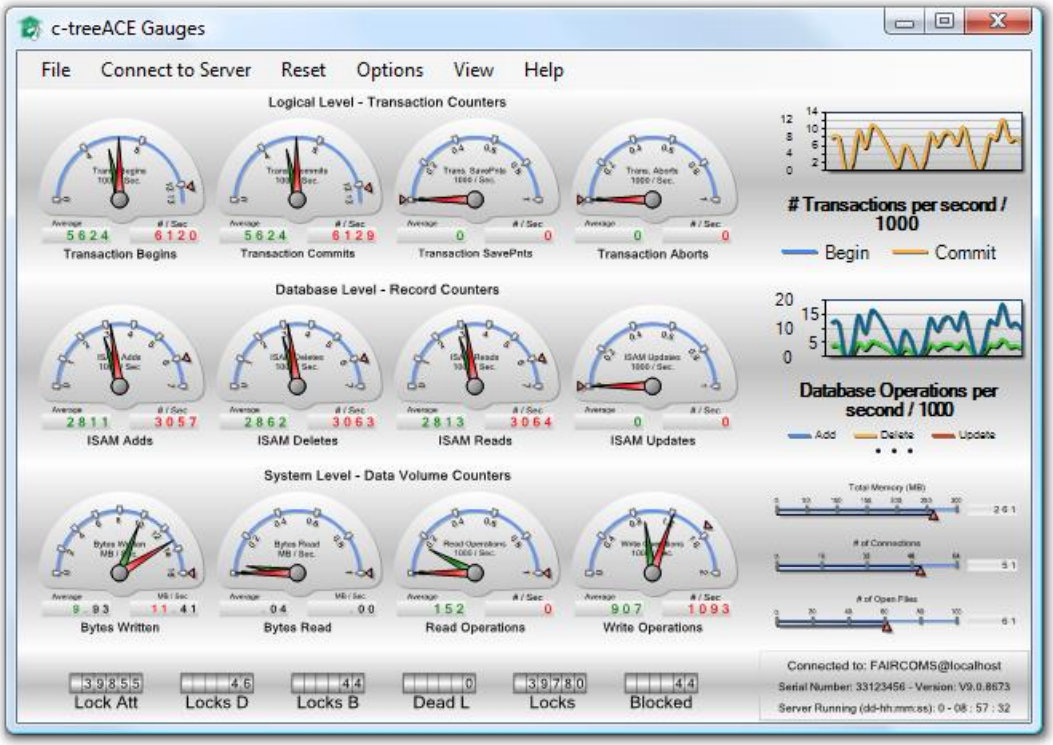
Updated in V11

- A freshened control layout for easier navigation
- Better execution plan displays
- Improved SQL script panels for editing using IF EXIST constructs
- Improved stability across all platforms

18.7 c-treeACE Gauges

c-treeACE Gauges tool presents a graphical display of many fundamental c-treeACE operating metrics. Multiple views delve into detailed performance measures for optimal server monitoring and tuning.

- **Two Views to Examine Real Time c-treeACE Operations**
- **Cumulative and Average Values Maintained for Comparison**

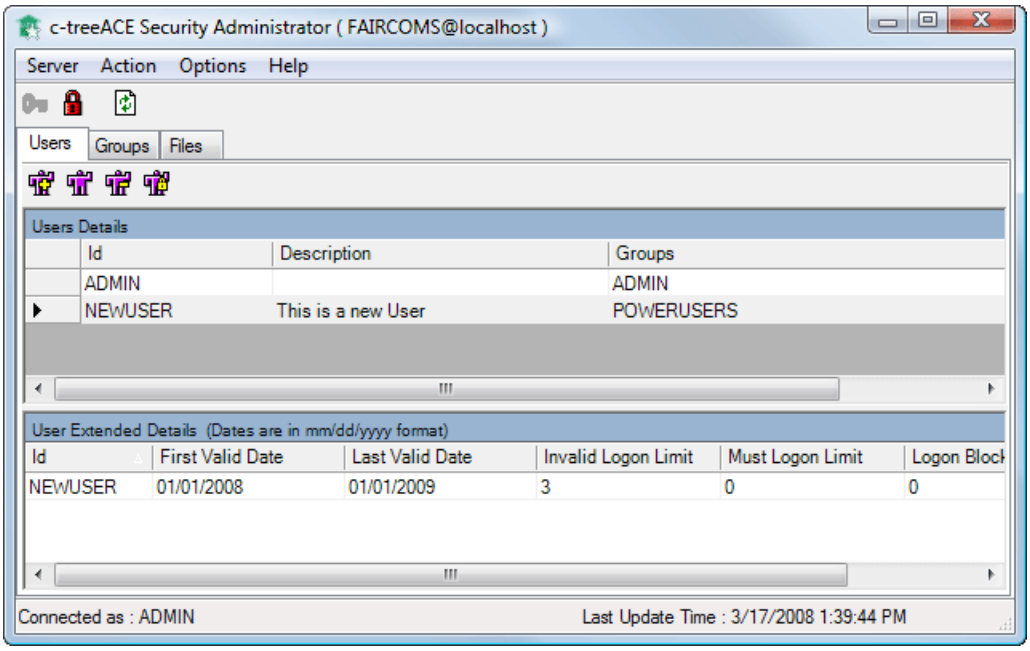


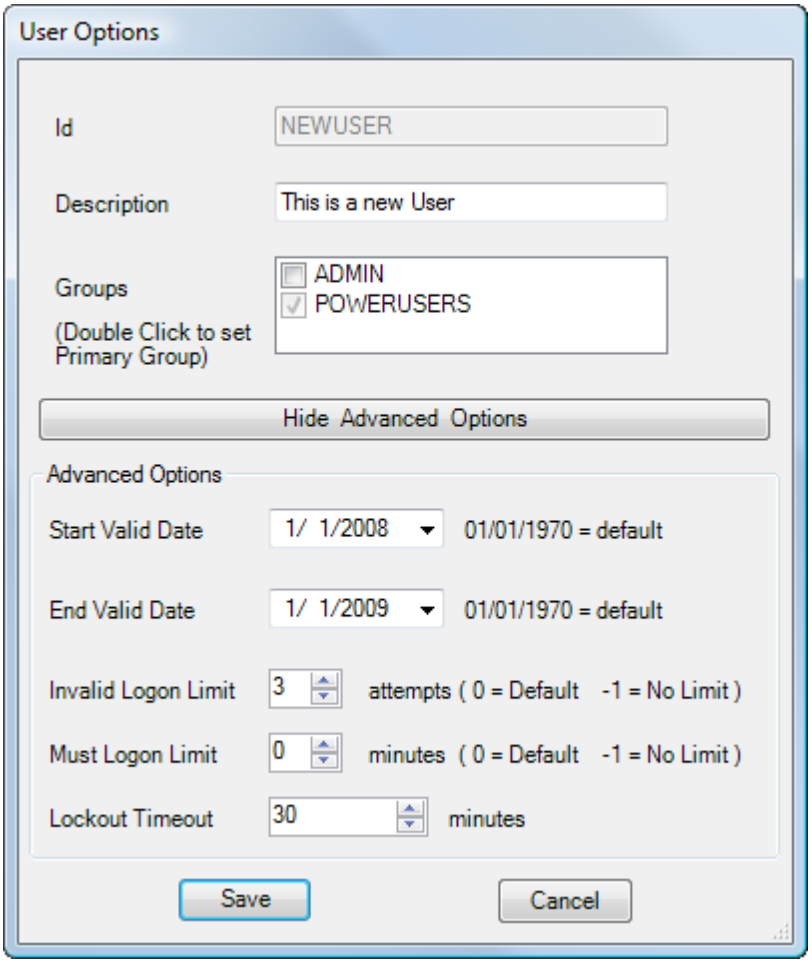

c-treeACE Gauges
File
Connect to Server
Reset
Options
View
Help
Transaction Counters (Logical Level)
Begins
Commits
Aborts
SavePnts
Record Counters (Database Level)
Reads
Adds
Updates
Deletes
Data Volume Counters (System Level)
Bytes Read
Read Operations
Bytes Written
Write Operations
Communication Counters (c-tree Level)
Comm. Bytes Read
Comm. Read Ops.
Comm. Bytes Written
Comm. Write Ops.
Log Operation Counters (System Level)
Log Write Ops.
Log Bytes Written
Log Read Ops.
Log Bytes Read
Log Flush Ops.
Log Ext. Ops.
Data/Index Caches (c-tree Level)
Data Buffer Reqst
Index Buffer Reqst
% Data Cache Used
% Index Cache Used
% Data Cache Hits
% Index Cache Hits
Sync/Flush Operations (System Level)
c-tree File Sync
Disk Sync
Tran Log Sync
Tran Logidx Sync
Begin Tran Flush
Comm/Abort Flush
LOCK Operations (System Level)
Locks Held
Blocked Relsd.
Locks Blocked
Lock Attempts
Locks Freed
Locks Denied

18.8 c-treeACE Security Administrator

The Security Admin tool provides an easy way to manage c-treeACE user and groups. Applications moving to c-treeACE SQL and ODBC are quickly finding that segmenting database access among multiple users becomes a challenging task. With new SQL group management, this becomes a much more powerful way to assign access rights to your users and the Security Admin tool only adds to your ease of management.

- **Add, Delete and Modify Users**
- **Change User Passwords**
- **Create, Modify and Delete Groups**
- **Modify File Security Attributes**
- **Change File Passwords**





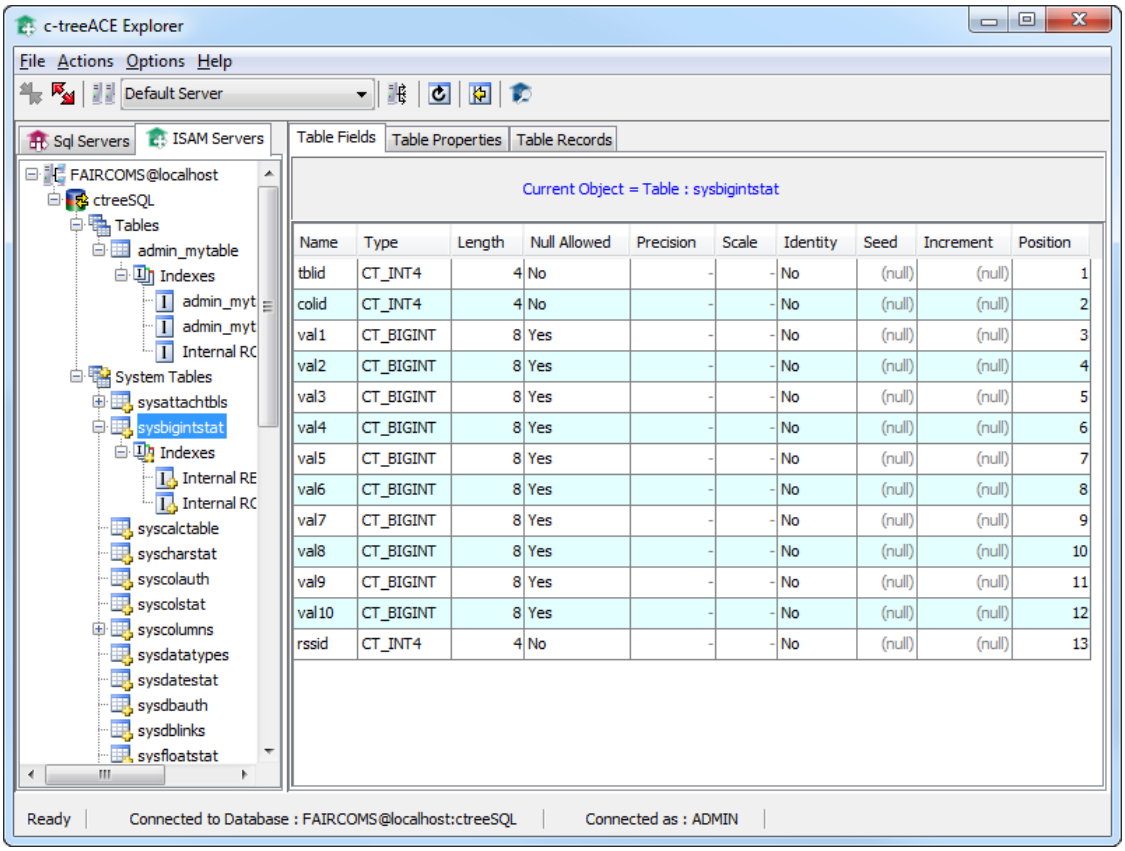
Updated in V11

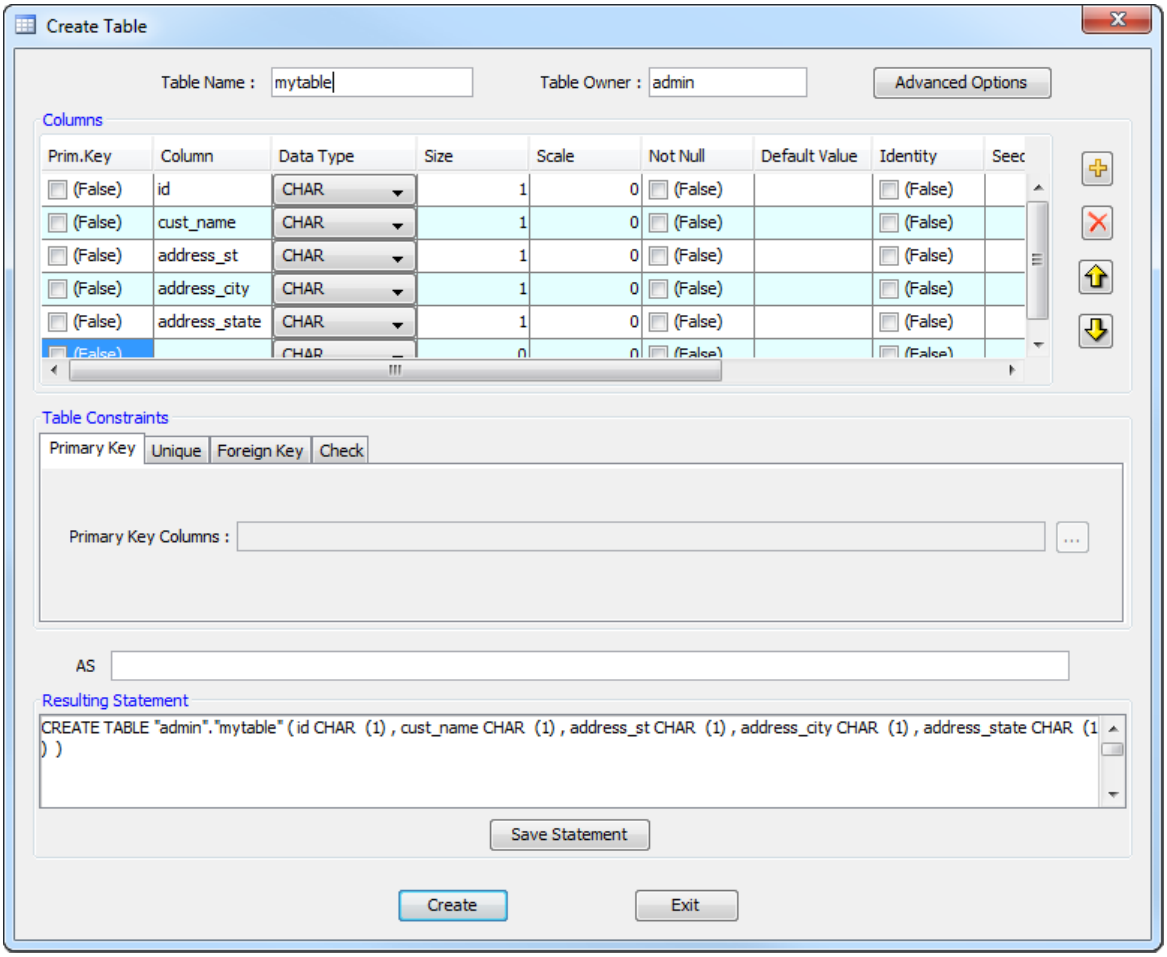
- Security attributes of all SQL related indexes are automatically adjusted when using the c-treeACE Security Administrator tool.
- Set permissions to a table and all its indexes at once.

18.9 c-treeACE Explorer

c-treeACE Explorer for Java allows cross platform database management. View and manage c-treeACE from Windows, Mac OSX and even Windows Unix/Linux environments with an installed Java platform.

- **Execute Custom c-treeACE SQL Statements**
- **Load, Edit and Run c-treeACE SQL Scripts Interactively**
- **View query execution plans to profile and examine complex queries for optimization analysis**
- **Create and Manage Databases**
- **Create, Alter and Drop Tables, Views, and Indexes**
- **Create and Drop Stored Procedures and Triggers**
- **Add Existing Files**
- **Browse Data**



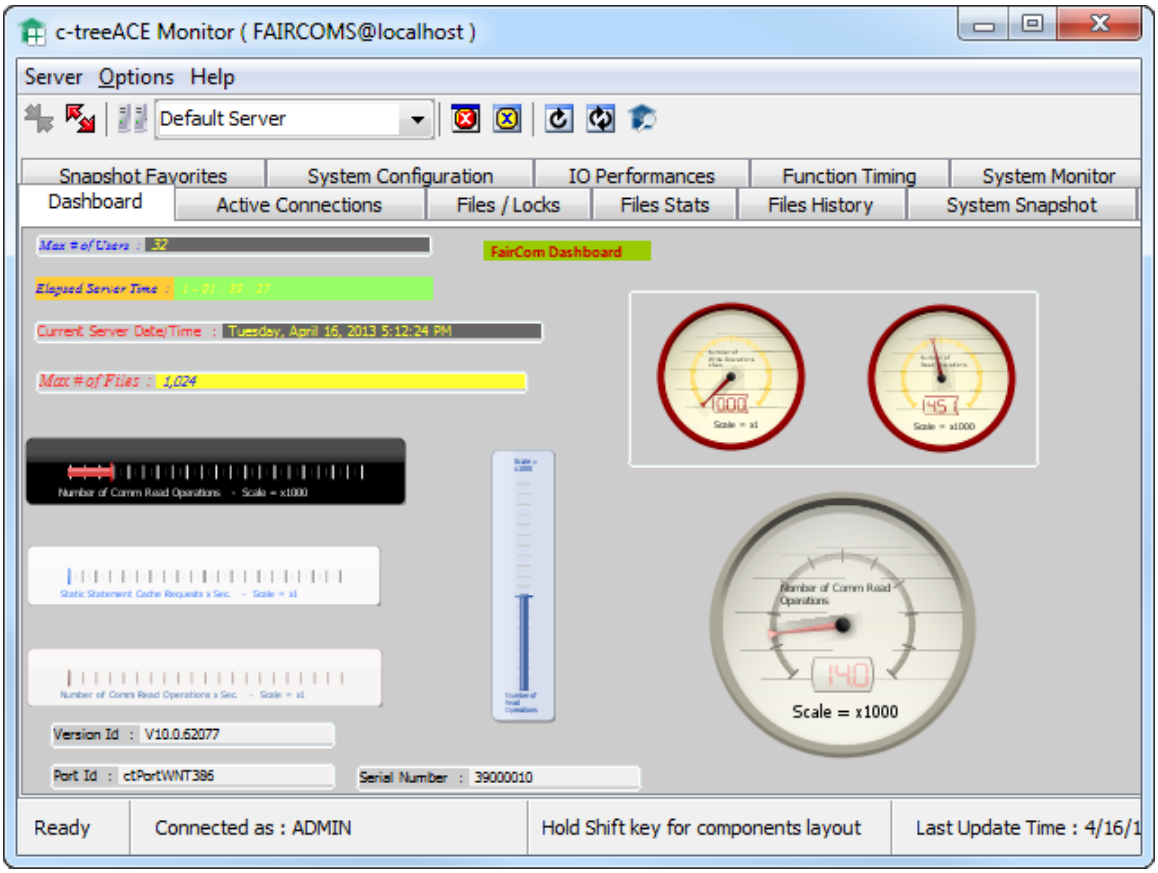


Updated in V11

- c-treeACE Explorer now supports different ways of encoding character sets beyond the UTF-8 that was originally supported. This support allows it to properly display "high-bit" ASCII (ASCII > 127) characters.
- The time spent during the fetch phase of query execution is now included in the total elapsed query time.
- Cosmetic changes were made in the area of the **Execution Plan** and the right panel. Performance was improved while making these changes.
- A new control was added allowing you to select the number of rows to display.

18.10 c-treeACE Monitor

Similar to the .NET version, this tool displays a wealth of performance metrics in many areas of c-treeACE operation.



18.11 Dr. c-tree

Designed as a low-level programmers' tool, DrCtree makes available all internal file structures. A great diagnostic tool during the development phase.

Tip: Check out the "Tree Walk" option to really dive into your c-tree index key layouts!

Figure 1: XCREblk tab page

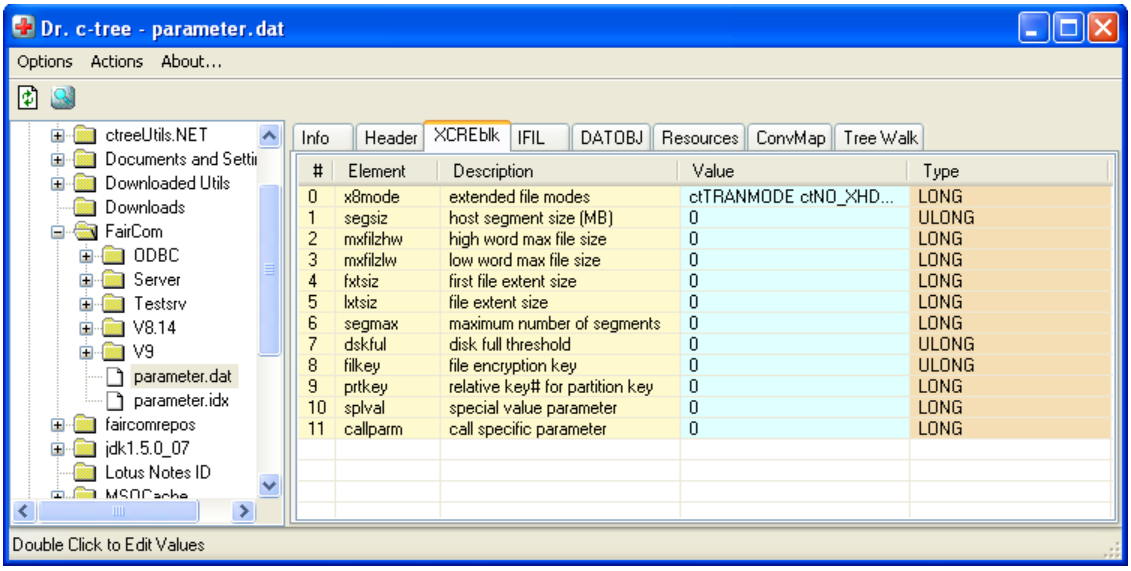
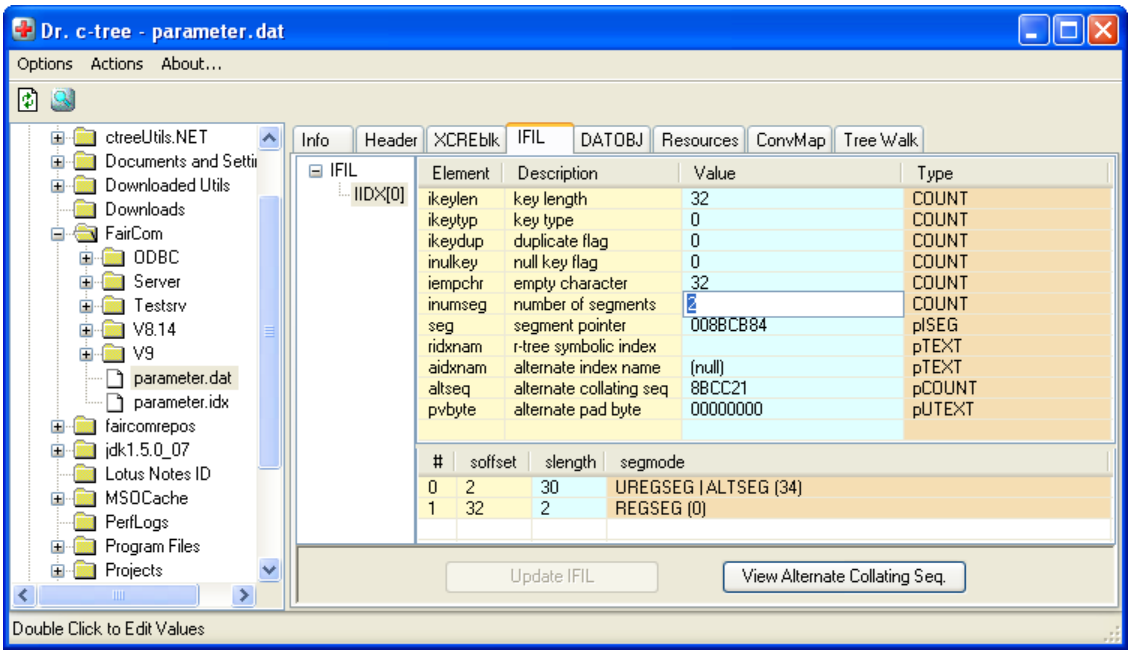
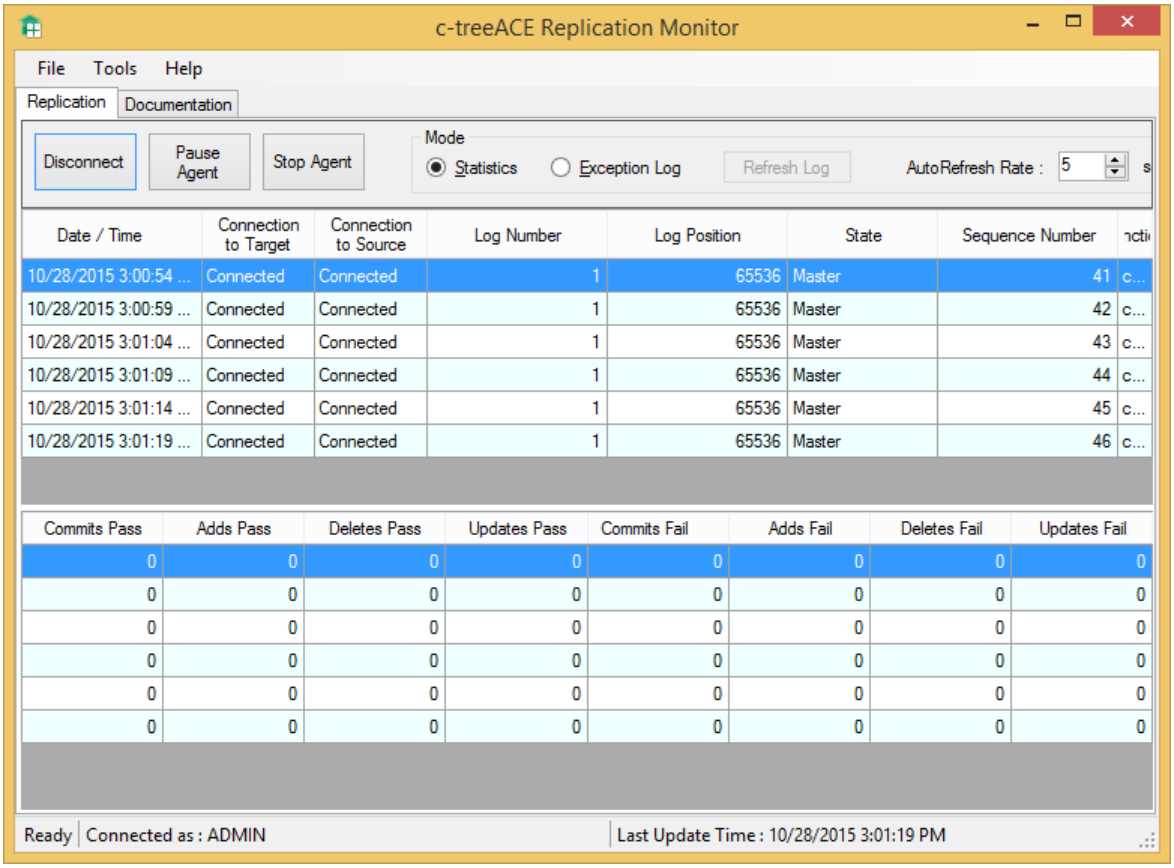


Figure 2: Altering IFIL entry



18.12 c-treeACE Replication Monitor

Monitor your c-treeACE Replication. Connect the Replication Monitor directly to your Replication Agent and view current status. Browse the exception log for failed transactions and associated messages.

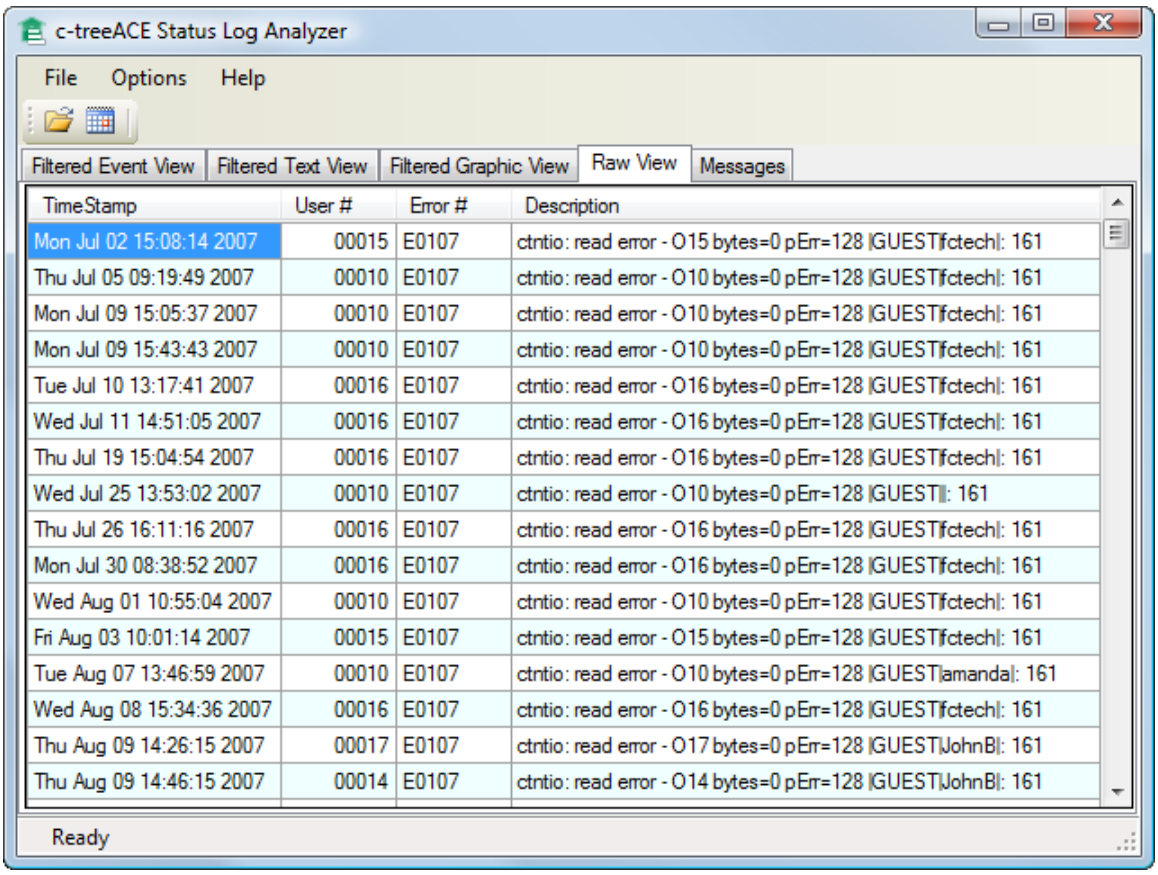


18.13 And More Tools...

You'll find many additional tools located in your /tools folder, both .NET and Java based for your platform preference.

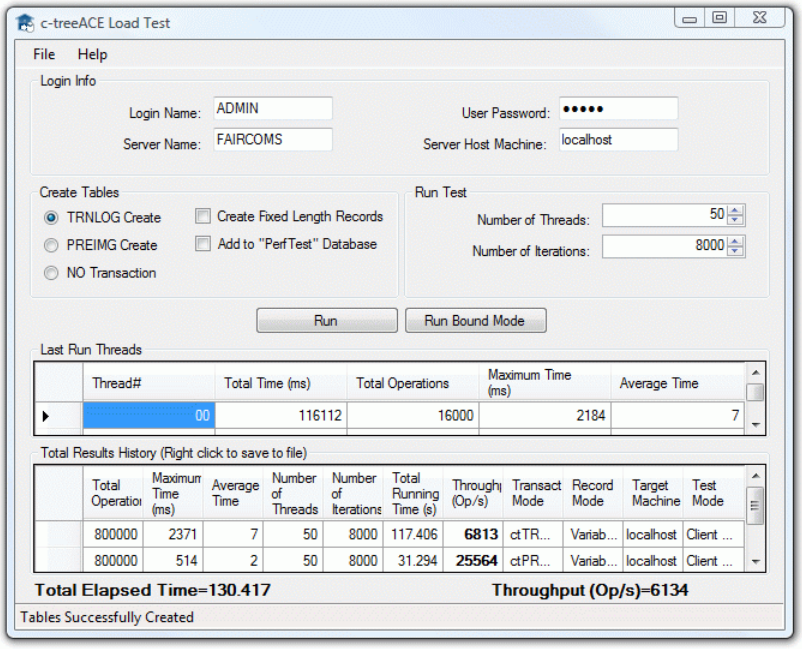
Log Analyzer

- **Sorted Entries by Operational Category**
- **Color Coded Messages with Critical Entries Highlighted in Red**



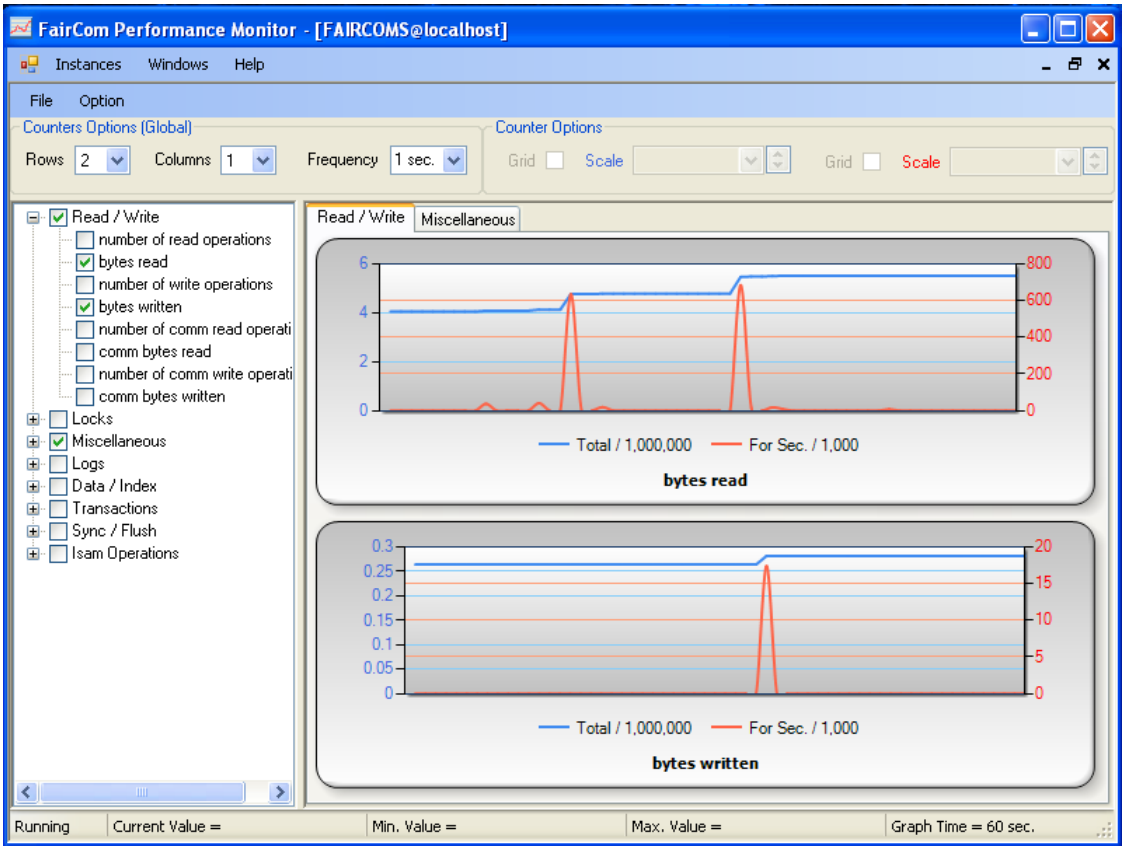
Benchmarking and Load Tests

- **Specify Multiple Threads of Operation**
- **Selectable Transaction Processing Levels**
- **Complete Display of Test Results for Each Run**



Performance Monitor

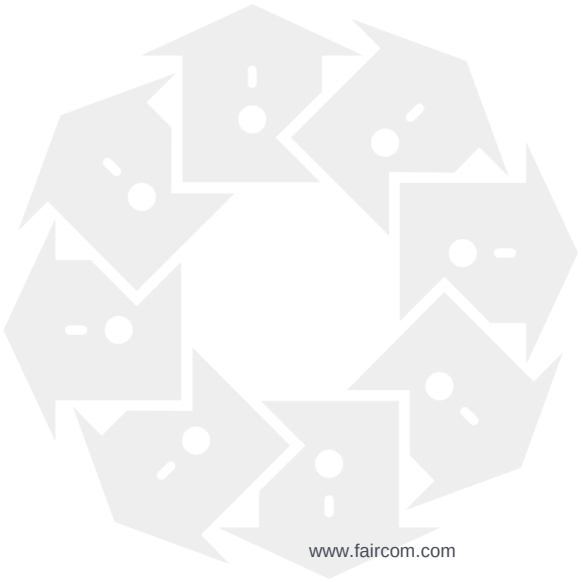
- **Graphically Display counters of your choice**
- **Real-time monitoring of c-treeACE activity**
- **A Windows Management Snap-in is also available with integrated counters monitoring.**



And many additional command-line (page 225) tool updates for V11 ...

19. Command-Line Utility Updates

Improvements in the command-line utilities give you expanded control of the c-treeACE data management environment.



19.1 Command-Line Tools for Administrators

Many c-treeACE activities are frequently scripted as part of larger application usage, especially initialization and maintenance tasks. c-treeACE provides a wide variety of tools for many common, and not so common tasks. These fall into several broad categories. You'll find tools for all of these, and many more, in your /tools/cmdline folder.

Note: Some tools interact directly against the server as a client utility. Others are standalone based, and do not require connection to a server. You'll find them organized into appropriate folders as such.

File Maintenance

- Information
- Compact
- Rebuild
- Transaction Control
- Verify Data and/or Index Integrity

Server Administration

- Statistics Monitoring
- Quiesce and File Block
- Remotely Stopping servers

Manage Access Control

- Managing Users and Groups
- Change User Passwords
- Manage Tamper Resistant configuration
- Manage Master Encryption Key

Backup and Restore

- Forward roll

Data and file Conversion

- Adding Advanced new Xtd8 Features
- Converting Original V4 files

SQL Management

- Interactive SQL Utility
- Create databases
- Import ISAM Tables into SQL
- Dump SQL Data
- Load Data into SQL

Check out the many need adjustments and corrections listed below.

19.2 FILESET host creation utility

To take advantage of FILESET functionality, it is necessary to have a host file created with *IFIL* and *DODA* attributes matching the files to be dynamically linked together.

A new utility has been added to create a FILESET host file based on a template table.

```
CreateFilesetHost -t template [-n host name] [-u user] [-p pwd] [-s server]
```

- *-u* - user name (default: admin)
- *-p* - password (default: ADMIN)
- *-s* - Server name (default: FAIRCOMS)
- *-t* - Existing data file to clone schema information
- *-n* - host table name (default: host_TEMPLATENAME)

After this file is created, import into c-treeACE SQL with the **ctsqlimp** SQL Import utility.

19.3 dfkctl - Deferred Index Maintenance Utility

Monitoring and Controlling Deferred Indexing

The **dfkctl** utility can be used to monitor and control deferred indexing threads.

Usage:

```
dfkctl  operation [-s svn] [-u uid] [-p upw] [-i int [cnt]] [-h frq]
                  [-d] [-m] [-t]
```

Operations:

- *-getstats* - Display deferred indexing statistics.
- *-getstate* - Display deferred indexing threads' state.
- *-getblstate* - Display background index load thread state.
- *-clearstats* - Reset deferred indexing statistics.
- *-pause* - Pause deferred indexing threads.
- *-resume* - Resume deferred indexing threads.
- *-queueidxload* - Queue background index load
- *-forcecommit* - Force the deferred indexer to treat its current transaction as a committed transaction. (For unusual cases when a deferred indexing thread hangs waiting for a transaction commit/abort that will never be signaled.)
- *-forceabort* - Force the deferred indexer to treat its current transaction as an aborted transaction. (An aborted transaction being recorded in the log would be unusual, but the option is included for completeness.)

Options:

- *-s svn* - c-tree Server name
- *-u uid* - User name

- *-p upw* - User password
- *-i int [cnt]* - Pause int seconds for optional cnt times
- *-h frq* - Print a description header every frq outputs
- *-d* - Show cache stats as delta
- *-m* - Show memory file stats when using -vaf report
- *-t* - Output timestamp with header

Examples of maintenance and monitoring with dfkctl

1. Display deferred indexing statistics:

```
# dfkctl -getstats -h 10

           succcessful operations:         failed operations:              skipped operations:
    qcnt    open     add  delete  update    open     add  delete  update     add  delete  update
       0   30004   93242       0       0       0       0       0       0    3000       0       0
       0   30004   93242       0       0       0       0       0       0    3000       0       0
```

2. Display deferred indexing thread state:

```
# dfkctl -getstate -h 10

  non-tran thread:        tran thread:
  current op    qcnt      current op  lognum  logpos
      readop       0          readop      19  18460903
      readop       0          readop      19  18460903
      readop       0          readop      19  18460903
      readop       0          readop      19  18460903
```

3. Clear deferred indexing statistics:

```
# dfkctl -clearstats

Successfully reset deferred indexing statistics.
```

4. Pause deferred indexing threads:

```
# dfkctl -pause

Successfully paused deferred indexing threads.
```

5. Resume deferred indexing threads:

```
# dfkctl -resume

Successfully resumed deferred indexing threads.
```

6. Queue a background index load:

```
# dfkctl -queueidxload <datafilename> <indexno> <indexno> ...
```

Where

- *<datafilename>* is the name of the data file
- *<indexno>* is the relative index number (one or more can be specified), with *indexno* of 1 representing the first index. Use a value of *all* to queue the loading of all indexes for the specified data file.

Queue loading of the first two indexes for data file *mark.dat*:

```
# dfkctl -queueidxload mark.dat 1 2 -u ADMIN -p ADMIN -s FAIRCOMS
```

Queue loading of all of the indexes for data file *mark.dat*:

```
# dfkctl -queueidxload mark.dat all -u ADMIN -p ADMIN -s FAIRCOMS
```

19.4 ctcompare - Database Comparison Tool

The Database Comparison utility, **ctcompare** is available in V11 and later. It allows you to compare two files (or two versions of the same file) or two tables. Options are provided that allow you to merge the differences into one of the files without affecting the other.

Usage:

```
ctcompare -s source -t target -n source name
        [-u source uid][-U target uid]
        [-p source pwd][-P target pwd][-N target name][-I name][-d]
        [-b source dir][-B target dir]
```

Where:

- *-bB* - base directory for session dictionary (only used with *-d*)
- *-d* - interpret name as a database name (default is table name)
- *-s* - source server name
- *-t* - target server name
- *-u* - source user name (default: admin)
- *-U* - target user name (default: admin)
- *-p* - source password
- *-P* - target password
- *-n* - source file name
- *-N* - target file name (default is the source name)
- *-x* - printf hex offsets
- *-I* - Use r-tree symbolic index 'name' as record identifier

The following parameters are used to apply differences from the source to the target. The source will remain unchanged. When determining the differences between the two files, records are identified based on their unique keys.

- *-A1* - Apply all differences (deletes extra records from target)
- *-A2* - Apply only missing records
- *-A3* - Apply missing records and updates

Example 1

Compare two files named *custmast.dat* on different servers, log differences:

```
ctcompare -s FAIRCOMS@localhost -t FAIRCOMS@otherhost -n custmast.dat -u admin -p ADMIN
```

Example 2

Compare two versions of the same file, and merge all differences to *custmast2*:

```
ctcompare -s FAIRCOMS -t FAIRCOMS -n .\dir1\custmast.dat -N .\dir2\custmast2.dat  -A1
```

Example 3

Compare all tables in two SQL/CTDB databases:

```
ctcompare -s FAIRCOMS -t FAIRCOMS@otherhost -n db1 -N db2 -d -u admin -p ADMIN
```

19.5 Rollback to New Restore Points with ctrdmp

In V11 and later, **ctrdmp** is able to rollback to a Restore Point. Restore Points permit server clients to establish quiet spots in the transaction log where there are no active transactions.

Prior to the V11 modifications, **ctrdmp** could either perform a dynamic dump recovery or rollback to a specified date and time. **ctrdmp** has been extended such that, as an alternative to specifying a date and time, the rollback script can provide the name of a Restore Point file.

A typical **ctrdmp** script file used for a rollback looks like:

```
!ROLLBACK
!DATE MM/DD/YYYY
!TIME HH:MM:SS
....
```

Now the script can be composed as follows:

```
!RP <Restore Point File Name>
....
```

The `Restore Point File Name` generated by the server is either of the following:

- *RSTPNT_NO_CHK.YYYYMMDD_HHMMSS.FCS* for a Lightweight Restore Point
- *RSTPNT_CHKPNT.YYYYMMDD_HHMMSS.FCS* for a Checkpoint Restore Point

Note that, as with the `!ROLLBACK` script keyword, the `!RP` keyword must be the first entry in the script file.

See also

- *ctrdmp - Dynamic Dump Recovery or System Rollback* (https://docs.faircom.com/doc/*ctserver/ctrdmp-util.htm*) • *ctfdmp - Forward Dump Utility* (https://docs.faircom.com/doc/*ctreeplus/ctfdmp-util.htm*)

19.6 Header Record Counts Output with ctinfo c-tree Information Utility

In V11 and later, the **ctinfo** utility outputs the header record for fixed-length files or the header key count for each logical index. For example:

```
Header Count
        4776 keys in index 0
        4776 keys in index 1
        4776 keys in index 2
```

19.7 Replication Actions Added to Transaction Control Utility cttrnmod

New replication actions have been added to the **cttrnmod** utility for flexible control of replication attributes.

- **cttrnmod** now displays replication state for a data file
- **cttrnmod** can change a file's replication state with the *repl* option

Note: Replication requires that the data file has a unique index and that the data and index files are using full (*ctTRNLOG*) transaction control.

Examples

1. Enable full transaction logging on files:

```
# cttrnmod set T -f files.txt -u ADMIN -p ADMIN -s FAIRCOMS

Setting transaction mode to ctTRNLOG for files listed in file
files.txt...

  Replicate  Tranmode  Filemode  Filename
  ---------  --------  --------  --------
         NO  ctTRNLOG    0x0032  ctreeSQL.dbs\admin_t.dat
             ctTRNLOG    0x0032  ctreeSQL.dbs\admin_t.idx
             ctTRNLOG    0x0032  ctreeSQL.dbs\admin_t_ti.idx
```

Note the "Replicate" column for current replication state information.

2. Enable replication on files:

```
# cttrnmod set repl=on -f files.txt -u ADMIN -p ADMIN -s FAIRCOMS
Enabling replication for files listed in file files.txt...

  Replicate  Tranmode  Filemode  Filename
  ---------  --------  --------  --------
        YES  ctTRNLOG    0x0032  ctreeSQL.dbs\admin_t.dat
             ctTRNLOG    0x8032  ctreeSQL.dbs\admin_t.idx
             ctTRNLOG    0x8032  ctreeSQL.dbs\admin_t_ti.idx
```

Note: If **cttrnmod** is used to disable full transaction logging for a file, it also disables replication for that file.

19.8 ctTRANMODE Control Added to Transaction Control Utility cttrnmod

When using the Transaction Control utility, **cttrnmod**, to disable transaction support on a file with extended file mode *ctTRANMODE*, the utility could report that after successfully disabling *ctTRNLOG*, the file still has *ctTRNLOG* set. This is expected for a file with the *ctTRANMODE* bit set when using a TRANPROC c-tree application.

cttrnmod has been updated to disable *ctTRANMODE* and *ctPIMGMODE* bits when it sets a file to no-transaction support. It was also modified to support explicitly enabling or disabling one of these bits (depending on the file mode that is in effect at the time).

19.9 Replication Debug Utility Timestamps Displayed in Local Time Format

The **ctrepd** utility now supports an option to display timestamps in human readable (local time) format: *-showlocaltime*.

Example output:

```
opcode = BEGTRAN
tranno = 16838303
tstamp = 1397160965: Thu Apr 10 15:16:05 2014
flags  = 0
lognum = 7986
logpos = 1088896 (0x00109d80)

opcode = ENDTRAN
tranno = 16838303
tstamp = 1397160966: Thu Apr 10 15:16:06 2014
flags  = 0
lognum = 7986
logpos = 1093717 (0x0010b055)

opcode = SUCTRAN
tranno = 16838303
flags  = 0
lognum = 7986
logpos = 1093951 (0x0010b13f)

Example output with -w option:

 log nbr    log pos  (in hex)    opcode       tranno fileid   tstamp                    flags
    7986    2248042 0x00224d6a  BEGTRAN     16841790      0   Thu Apr 10 15:20:02 2014      0
    7986    2248220 0x00224e1c  ENDTRAN     16841790      0   Thu Apr 10 15:20:02 2014      0
    7986    2248262 0x00224e46  SUCTRAN     16841790      0                                 0
```

19.10 Replication Debug Utility Options to Specify User Name and Password

The **ctrepd** utility now has options to specify the user name and password to use when connecting to the FairCom Server whose logs are being read. Use the *-u:<username>* and *-p:<password>* options. For example, if the user name and password are both `ADMIN`:

```
ctrepd 1 0 -u:ADMIN -p:ADMIN FAIRCOMS
```

19.11 Limit Replication Debug Utility Output to Specific Files

The **ctrepd** utility now supports an option to specify the data files whose activity it is to display. To use this feature, specify the *-lf:<filename>* option (lower case "lf" as in "limit file") when running **ctrepd**. *<filename>* is the name of a text file containing data file names, one per line, which can include wildcards. If a file name matches one of the file names specified in this file, **ctrepd** will display its activity.

Example:

Create the file *files.txt* containing these two lines:

```
*ctreesql.dbs\hum_b*.dat
*ctreesql.dbs\hum_a*.dat
```

Then run:

```
ctrepd 1 0 -w -n -lf:files.txt -f -m FAIRCOMS
```

ctrepd will only show the activity for these files.

Note: When using this option, the following entries are shown:

- Record add, delete, and update operations are shown.
- File open and close entries are *not* displayed because they are not of interest.
- Transaction begin and commit entries are *not* displayed.

ENDTRAN entries for transactions that have one or more operations for files in the filter list are displayed because they contain the time stamp at which the changes were committed.

19.12 New File Verification Utilities - ctflvrfy.exe, ctvfyfil.exe, ctvfyidx.exe

Exceptional situations exist where files may be left in inconsistent states. It was requested to be able to more formally, and comprehensively, verify data and index file integrity. The backbone of this cross-file integrity check are new API calls available in the c-tree SDK. These verification APIs allow cross checking files in both directions: verifying all index entries have a matching data file record, as well as ensuring each data file record has consistent index key values.

Along with the introduction of these new c-tree API calls, several utilities were introduced with this functionality available for ease of routine use.

ctvfyfil.exe and **ctvfyidx.exe** utilities (described below) have been added to the client Admin area.

The following utilities have been added to the standalone low-level package:

ctflvrfy.exe

Usage

```
ctflvrfy <filename.idx>
```

Description

The Index Verify utility, **ctflvrfy**, takes the index name as the parameter and calls the **ctVERIFY()** and **chkidx()** functions to allow the user to verify the index, and optionally inspect it at a low-level.

```
>ctflvrfy custmaster.idx

Verifying file [custmaster.idx]
Retrying open with sect of [8]
Retrying open with sect of [12]
Retrying open with sect of [16]
Retrying open with sect of [20]
Retrying open with sect of [24]
Retrying open with sect of [28]
Retrying open with sect of [32]
Retrying open with sect of [36]
Retrying open with sect of [40]
Retrying open with sect of [44]
Retrying open with sect of [48]
Retrying open with sect of [52]
Retrying open with sect of [56]
Retrying open with sect of [60]
Retrying open with sect of [64]
Verifying the host index

Verifying index delete stack...

Verifying index links...

Verifying index leaf nodes...

Index page scan finds entries=75  header=75
Index nodes per level of tree structure - [0: 1]
( 2 resource pages )
Internal Index Verify: SUCCESSFUL
The return of ctVERIFY = 0,  75 KEYS FOUND
Verifying member index #1

Verifying index delete stack...

Verifying index links...

Verifying index leaf nodes...

Index page scan finds entries=75  header=75
Index nodes per level of tree structure - [0: 1]
( 2 resource pages )
Internal Index Verify: SUCCESSFUL
The return of ctVERIFY = 0,  75 KEYS FOUND
Verifying member index #2

Verifying index delete stack...

Verifying index links...

Verifying index leaf nodes...
```

```
Index page scan finds entries=75  header=75
Index nodes per level of tree structure - [0: 1]
( 2 resource pages )
Internal Index Verify: SUCCESSFUL
The return of ctVERIFY = 0,  75 KEYS FOUND
Verifying member index #3

Verifying index delete stack...

Verifying index links...

Verifying index leaf nodes...

Index page scan finds entries=75  header=75
Index nodes per level of tree structure - [0: 1]
( 2 resource pages )
Internal Index Verify: SUCCESSFUL
The return of ctVERIFY = 0,  75 KEYS FOUND
Verifying member index #4

Verifying index delete stack...

Verifying index links...

Verifying index leaf nodes...

Index page scan finds entries=75  header=75
Index nodes per level of tree structure - [0: 1]
( 2 resource pages )
Internal Index Verify: SUCCESSFUL
The return of ctVERIFY = 0,  75 KEYS FOUND
Verifying member index #5

Verifying index delete stack...

Verifying index links...

Verifying index leaf nodes...

Index page scan finds entries=38  header=38
Index nodes per level of tree structure - [0: 1]
( 2 resource pages )
Internal Index Verify: SUCCESSFUL
The return of ctVERIFY = 0,  38 KEYS FOUND
Verifying member index #6

Internal Index Verify: no tree structure

The return of ctVERIFY = 0,  0 KEYS FOUND
Verifying member index #7

Verifying index delete stack...
```

```
Verifying index links...

Verifying index leaf nodes...

Index page scan finds entries=75  header=75
Index nodes per level of tree structure - [0: 1]
( 2 resource pages )
Internal Index Verify: SUCCESSFUL
The return of ctVERIFY = 0,  75 KEYS FOUND
Verifying member index #8

Verifying index delete stack...

Verifying index links...

Verifying index leaf nodes...

Index page scan finds entries=16  header=16
Index nodes per level of tree structure - [0: 1]
( 2 resource pages )
Internal Index Verify: SUCCESSFUL
The return of ctVERIFY = 0,  16 KEYS FOUND
Verifying member index #9

Verifying index delete stack...

Verifying index links...

Verifying index leaf nodes...

Index page scan finds entries=75  header=75
Index nodes per level of tree structure - [0: 1]
( 2 resource pages )
Internal Index Verify: SUCCESSFUL
The return of ctVERIFY = 0,  75 KEYS FOUND
Verifying member index #10

Verifying index delete stack...

Verifying index links...

Verifying index leaf nodes...

Index page scan finds entries=75  header=75
Index nodes per level of tree structure - [0: 1]
( 2 resource pages )
Internal Index Verify: SUCCESSFUL
The return of ctVERIFY = 0,  75 KEYS FOUND
Verify complete......
Do you wish to perform a low-level tree walk? (Y/N):  n
```

ctvfyfil.exe

The **ctvfyfil** utility calls the **ctVerifyFile()** (, **/doc/ctreeplus/ctverifyfile.htm**) function. (See function for details.) The utility can be run in standalone and in client/server mode.

Usage

```
ctvfyfil [<option> ...] <file name>
```

where *<option>* is one or more of the following:

- *<page size>* - Use the specified page size
- *-chkdat* - Read data file only (default)
- *-chkdatkeys* - Read data file and check keys in index
- *-chkdeletedspace* - Check deleted space
- *-chkidx* - Read index file
- *-chkidxrec* - Read index file and check records in data file
- *-chkidxint* - Additional index checks
- *-excl* - Open the file in exclusive mode
- *-int* - Interactive mode: stop on each error
- *-index=<N>* - Check only the *N*th index (N=1,2,...)
- *-local* - causes the utility to use a standalone (local side) connection instead of a client connection when linked with a LOCLIB library.

The example below shows the utility run on a file called *mark.dat* with the page size set to 8192:

```
ctvfyfil -8192 mark.dat
```

ctvfyidx.exe

The **ctvfyidx** utility uses the **ctVERIFYidx()** function to check the integrity of an index file. The client version of the **ctvfyidx** utility supports the command-line options listed below.

Usage

```
ctvfyidx [<option> ...] [-u <userid>] [-p <password>] [-s <servername] <file name> [<member #>]
```

where *<option>* is one or more of the following:

- *-excl* - Open the file in exclusive mode
- *-delstk* - Check delete stack (on by default)
- *-link* - Check links (on by default)
- *-leaf* - Check leaf nodes (on by default)
- *-local* - causes the utility to use a standalone (local side) connection instead of a client connection when linked with a LOCLIB library.
- *-chkkey* - Check keys in leaf nodes

The optional parameter *-page size* equals sector size * 128 (third parameter in **InitCtree()**). If page size is not entered, a default value of 16 will be used. *filename* specifies the index file targeted for analysis. The *member #* refers to the index member number. A physical index file can contain one or more indexes. Each index has a member number (0, 1, 2, 3, etc.). For example, the sample index file *custordr.idx* provided with the FairCom ODBC Driver contains a total of two indexes. Depending on whether you specify 0 or 1 you will be looking at either the order number index or the customer number index. *rflg* represents an optional recovery flag switch and is only applicable when compiled with *TRANPROC*. Any character will enable *rflg*, which will result in c-tree skipping automatic recovery.

The **ctvfyidx** utility defaults to ctREADFIL. It uses ctEXCLUSIVE when the *-excl* option is specified. A ctREADFIL open will fail with error **12** and sysiocod -8 if any connection has the file open for write access.

Example

Below is an example of launching **ctvfyidx** along with output showing the results of the index verification.

```
# ctvfyidx -2048 custmast.idx 0

Index page scan finds entries=4  header=4
Index nodes per level of tree structure - [0: 1]
Internal Index Verify: SUCCESSFUL
```

19.13 Advanced encryption master key store encrypted at system level on Windows

c-treeACE supports creating an advanced encryption master key store encrypted at the system level on Windows. Prior to this revision, the encrypted master key store file created by the **ctcpvf** utility on Windows could only be decrypted by the user account that created the file. This made it difficult to set up a Windows service that is using the LocalSystem account to be able to read the encrypted master key store file. (The **ctcpvf** utility had to be run as LocalSystem when creating the master key store.)

An option has been added to the **ctcpvf** utility to create the encrypted store using system-level encryption, meaning that any user account on the system can decrypt the file. Use the **ctcpvf** utility's *-syslevel* option to use this feature. Example:

```
ctcpvf -k mymasterkey -s ctsrvr.fkf -syslevel
```

This option has been added to the **ctadmn** utility's "Change advanced encryption master password" option. Example:

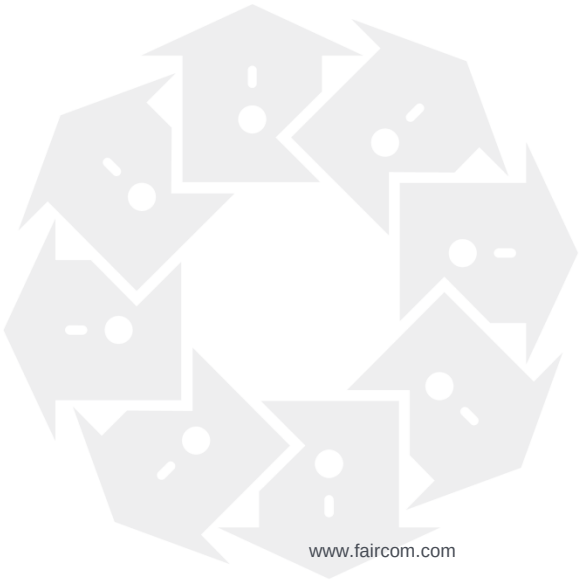
```
Enter the name of the filename list file >> files.txt

Enter the current advanced encryption master password >> ***************

Enter the new advanced encryption master password >> *****************

Please confirm the new master password by entering it again:

Enter the new advanced encryption master password >> *****************

Enter the encryption level [U]ser or [S]ystem for the encrypted store >> u


Changing master password for the specified files...


Successfully changed the advanced encryption master password.
```

See *ctadmn.c* for an example showing how to call the **SECURITY()** function with mode of SEC_CHANGE_ADVENC_PASSWD to change the master key. If you want to create the master key encrypted store using the system-level encryption option, OR in the ctENCMODsysl bit to the options field of the ctENCMOD structure whose address you pass to **SECURITY()**.

Note: This support was added on the Windows platform only.

20. Critical Production Updates

FairCom firmly believes in minimizing changes that impact systems that are in production. However, from time to time it is necessary to make such changes to ensure proper operation of your database. Be sure to familiarize yourself with these important changes that may affect systems that are already in production.



20.1 Corrected Unhandled Exception When File Password Included in Open File Call

When using the c-treeACE Administration utility, **ctadmn**, to connect to FairCom Server, if a file password was specified for *FAIRCOM.FCS*, FairCom Server terminated with an unhandled exception. This could also happen on any call to open a file that specified a non-empty file password on a system that requires properly-aligned memory accesses. The logic has been modified to correct this problem.

20.2 Corrected Read and Write Errors after Connection Termination

Using **ctadmn** or c-treeACE Monitor to terminate a SQL or ISAM connection to c-tree Server that is using the TCP/IP communication protocol can cause errors reading and writing files or an unhandled exception when reading data from the socket. *CTSTATUS.FCS* may show errors such as the following:

```
 - User# 00285 External kill request posted against user #89
 |ODBCUSER|SQL:LIVE|
 - User# 00178 ctsave failed: system code = 9  lc = 38  fd = 946
 - User# 00178 ./live.dbs/appkcomp.dat
 - User# 00178 Error on close file. locale:20  sysiocod:9  uerr_cod: 24
 - User# 00178 ./live.dbs/appkcomp.dat
 - User# 00047 WRITE_ERR: ./live.dbs/rlavllog.dat at 0:0x  sysiocod=9 bufsiz=8192 bytes written=0[0]
ioLoc=0: 37
 - User# 00047 FAILED TRAN IO: ctwrtbuf...: 37
 - User# 00047 ./live.dbs/rlavllog.dat: 37
 - User# 00047 Dumped stack for server process 29098172, log=1, loc=60, rc=0
 - User# 00047 O47 M1 L60 F2720 P0x (recur #1) (uerr_cod=37)
```

The logic has been modified to correct these errors.

20.3 Prevent FairCom Server WRITE_ERR Termination with Open Transactions Aborted by Quiesce

The FairCom Server was found to shut down due to a write error on a transaction-controlled file in the following precise set of circumstances:

1. A client begins a transaction and updates a transaction-controlled file.
2. While the transaction is still active, the server is quiesced using the **ctquiet** utility's "full consistency" option.
3. The quiesce aborts the client's transaction.
4. If the client then closes its connection while the quiesce is still active, the server shuts down due to a write error on the client's file.

FairCom Server logged the following error messages to *CTSTATUS.FCS* and shut down:

```
Thu Feb 13 09:34:05 2014
 - User# 00014 WRITE_ERR: vcusti at 0:4000x  sysiocod=6 bufsiz=8192 bytes written=0[0]  ioLoc=0: 37
Thu Feb 13 09:34:09 2014
 - User# 00014 FAILED TRAN IO: ctwrtbuf...: 37
Thu Feb 13 09:34:09 2014
 - User# 00014 vcusti: 37
Thu Feb 13 09:34:11 2014
 - User# 00014 Dumped stack for server process 67504, log=1, loc=60, rc=0
Thu Feb 13 09:34:11 2014
 - User# 00014 ctQUIET: unblocking call with action: 9e00x
Thu Feb 13 09:34:11 2014
 - User# 00014 ctQUIET: end of unblocking call
Thu Feb 13 09:34:13 2014
 - User# 00014 O14 M1 L60 F23 P4000x (recur #1) (uerr_cod=37)
```

The logic has been modified to correct this situation.

20.4 Corrected Unexpected FairCom Server Internal Error 7495 Crash

In an unusual situation, FairCom Server could crash during commit or abort when a memory allocation occurred. There were two possible symptoms depending on the platform:

1. On a high/low system: FairCom Server terminated with an unhandled exception.
2. On a low/high system: FairCom Server terminated with internal error **7495**.

The memory allocation failure was already detected and handled, however, its handling caused an unexpected internal state in the server, which later caused the observed symptom.

The logic has been modified to correct this situation.

20.5 Prevent c-tree Server Unhandled Exception During Update of Compressed Record

c-tree Server terminated with an unhandled exception when updating a compressed record in a *HUGE* transaction-controlled file (*ctTRNLOG* or *ctPREIMG*) at the logical end of file after turning off ISAM key buffers if the transaction had a previous update to the record followed by a savepoint.

The significant factors were:

- the record was compressed
- prior to the update where the crash occurred, the same transaction updated the record and then established a savepoint

Compressed records no longer cause this exception.

20.6 Inconsistent FPUTFGET Header Locking for Non-HUGE Index Files and Variable-Length Data Files

A variety of errors were encountered in *FPUTFGET* mode when data was updated. The errors included **69**, **527**, **42**, **160**, **30**, **519**, and internal error **233**. For non-huge index files and variable-length data files, two connections could be updating a header at the same time, causing one's updates to be lost. This could cause the logical end of file value to be smaller than expected, causing space to be reused for index nodes or variable-length data records. The logic has been modified to correct these problems.

20.7 Avoid FairCom Server Termination with Internal Error 8987 When Using UNBUFFERED_IO Configuration Option

After enabling the UNBUFFERED_IO configuration option for a data file, FairCom Server terminated with internal error **8987**. The logic has been corrected to eliminate this occurrence.

20.8 Prevent Unhandled Exception When a Single Connection Opens a File More than 1024 Times

FairCom Server terminated with an unhandled exception after a single connection opened a file more than 1024 times. The logic has been modified so that attempting to open a file more than 1024 times in a single connection will now fail with error code **COFM_ERR** (1103) and will preserve the proper co-file state. If a `-1` co-file link is found when closing the file (which is now unexpected), the code avoids using the invalid co-file link.

20.9 Corrected Prime Cache Thread Unhandled Exception When Opening File Pending Delete

An unhandled exception occurred when a prime cache thread opened a file that was pending delete. The logic has been modified so that the server returns an error in that situation.

20.10 Corrected Errors When Changing a Temporary Index Condition

If a condition was set on a temporary index by one client and then another client changed the condition, an unhandled exception, an internal error, or an **NKEY_ERR** may have occurred. An example of the internal error:

```
Fri Apr 11 16:08:15 2014
 - User# 00217 7491: Memory return out of bounds...
Fri Apr 11 16:08:15 2014
 - User# 00217 Limit:8000000x Return:899c5360x
Fri Apr 11 16:08:15 2014
 - User# 00217 Unexpected internal c-tree(R) error #7491 (uerr_cod=0)
```

The logic has been modified to correct this.

20.11 Deadlock Corrected in Data Cache Retrieval Function

A deadlock was found in a data cache retrieval function. This could cause c-treeACE Server to hang during normal operation on a call to request a file's mutex when retrieving data from the data cache. (This condition only occurred in c-treeACE Server v10.3.0.) The logic has been corrected to eliminate this problem.

20.12 Unhandled Exception When Accessing Pruned Memory Index Node

FairCom Server caused an unhandled exception when accessing a memory index node that had been pruned from the tree. The logic has been corrected to eliminate this problem.

21. Notable Compatibility Changes

This section lists important known compatibility changes from prior releases.

It is important to review these issues and ensure that your application continues to correctly function after upgrading to this latest release, V2.

21.1 FairCom Server - Change defaults for V11 release

FairCom Server now uses the following default values for these configuration options when they are not specified in *ctsrvr.cfg*:

```
CHECKPOINT_FLUSH        17
LOG_SPACE               120 MB
CHECKPOINT_INTERVAL     10 MB
LOG_TEMPLATE            2
COMMIT_DELAY  2 (1 on Linux)

; Data and index cache size
DAT_MEMORY              100 MB
IDX_MEMORY              100 MB

; Sort memory for index rebuild
SORT_MEMORY             100 MB

; Maximum status log size of 32 MB, keeping one prior copy
CTSTATUS_SIZE           -32000000
```

These options are new to V11, and these are their defaults:

```
; Flush updates for transaction files to file system as soon as possible in the background
TRAN_DATA_FLUSH_SEC     60
TRAN_INDEX_FLUSH_SEC    60

; Flush updates for non-tran files to file system as soon as possible in the background
NONTRAN_DATA_FLUSH_SEC  IMMEDIATE
NONTRAN_INDEX_FLUSH_SEC IMMEDIATE
```

The following options have been added as convenience options. However, you should review them as they may change behavior in your particular installations in subtle ways.

```
; Limit JVM memory
SETENV                  DH_JVM_OPTION_STRINGS=-Xms100m -Xmx300m

; Suppress dynamic dump logging of backed up file names to CTSTATUS.FCS
CTSTATUS_MASK           DYNAMIC_DUMP_FILES

; Log final SNAPSHOT stats to SNAPSHOT.FCS on shutdown for baseline metrics
DIAGNOSTICS             SNAPSHOT_SHUTDOWN
```

21.2 Correct Error Messages Now Returned by fc_create_user Procedure

Calling **fc_create_user** could have resulted in several unusual error messages, due to an overlap of the SA_ADMIN API and c-treeACE SQL error codes. For example, calling this procedure with an empty user name previously returned error code **-18008**, which was reported as message, "CT - A partitioned file can only have one open instance at a time per connection." The SA_ADMIN return codes used by this function have been replaced with c-tree error codes. In the example case, the error code is now correctly returned as **-17450** with message "CT - Invalid user id."

21.3 SQL - Changed error message for error -20139

The error message for error **-20139** has been updated to be more meaningful. The message now states:

```
error(-20139): Index on long fields not supported
```

21.4 SQL - BINARY fields not padded with 0x00

This is a behavior change that affects how the values of BINARY fields are stored in V11 and later.

Prior to V11, the server was not padding BINARY fields with 0x00 when the values were less than the defined length. Because of this, both the values stored on disk and the values returned may have been shorter than the defined length (basically BINARY and VARBINARY behaved in the same manner). This behavior has been corrected in V11 and later so that BINARY fields are properly padded with 0x00 both on disk and when retrieved. Existing data is not touched on disk, however the values returned in SQL are now padded.

The previous behavior can be restored by using the keyword `SQL_OPTION NO_BINARY_PAD` in *ctsrvr.cfg.*

This is only documented for a select few applications that may be impacted by the V11 change in behavior. Consult FairCom support should have you concerns about your current binary data prior to V11.

USE OF THIS OPTION WILL LIKELY RESULT IN UNEXPECTED BINARY FIELD BEHAVIOR.

21.5 SQL - Binary Literals

In V11 and later, `'1234'` is interpreted as a character string. In V10, this value was treated as a hexadecimal string (arbitrary character strings were not supported in V10).

Example:

```
INSERT INTO mybinary values('1234');
INSERT INTO mybinary values(x'31323334')
```

In V11 and later, both the above inserts insert the same value. In V10 they would both be treated as hex strings.

21.6 Automatic FairCom DB API Batch Buffer Resize

The **ctdbSetBatch()** function takes a parameter *buffeLen* which is the size of the buffer used internally by FairCom DB API to handle batch operations.

```
CTDBRET ctdbDECL ctdbSetBatch(CTHANDLE Handle, CTBATCH_MODE mode, VRLEN targetLen, VRLEN bufferLen)
```

A value of 0 for this parameter was an indication that the default value size should be used. The default buffer size is calculated as the size of the fixed portion of the record multiplied by 128.

In this release and later, when *bufferLen* is set to 0, the default buffer size is calculated as described above, and logic is activated to perform automatic buffer resize if the buffer is not large enough to contain one record.

When *bufferLen* is not 0 and the buffer is not large enough to contain at least one record, the error **BTBZ_ERR** (429) is returned. In this case, it is possible to activate the logic to automatically resize the buffer by adding the new *CTBATCH_AUTORESIZE* FairCom DB API batch mode to the *mode* parameter.

21.7 Proper positioning of ctdbSeekRecord with active record sets

ctdbSeekRecord() logic was modified such that it positions into the set if a record handle has an active criteria set. Prior to this change, if a record handle had an active criteria set, **ctdbSeekRecord()** was not positioning into the set.

21.8 Server process exit code more informative

The process return code was reviewed and made to be more informative. Now, an external server stop request should return **0** on successful shutdown. A shutdown not triggered by external request will return a non-zero value.

21.9 Physical read of variable-length transaction controlled file skips records added by a third-party transaction not yet committed

A feature introduced in V8.14 affected the behavior of **NXTVREC** in physical order when it encountered the "record space" for a new record that had not been committed and was written in a space that was not being reused. Instead of reporting a **VFLG_ERR** (error 158), that new feature skips the uncommitted "record space" (unless the reading is by the transactor, which would see the uncommitted record).

Details

The change affects how variable-length records are internally marked during transaction processing of pre-image space. The behavior prior to V8.14 marked the record header in a way that was considered invalid, causing a **VFLG_ERR** (158) error. The newer behavior sees the record as a deleted record (actually a pending insert) and skips to the next record as in a "read committed" transaction isolation.

This change is limited to the internal handling of header marks for newly added variable-length records.

Indexed files are not affected by these changes because pending key inserts are handled differently under transaction control.

These changes do not include changes to the physical files, record structures on disk, or other transaction control.

Reverting Back to the Old Behavior

These changes can be reverted back to the original (prior to V8.14) behavior using the keyword:

`COMPATIBILITY NO_INIT_VSPACE` (https://docs.faircom.com/doc/`ctserver/57570.htm`)

Changes in the Latest Revision

The changes introduced in the V11 release address the issues with the earlier change as follows:

If the reader has requested acquiring locks on the records that it reads, the physical read acquires a lock on that record and respects the lock before proceeding (earlier the lock was not respected) producing one of the following outcomes:

If...	Then...
Locking the record fails with error 42 (**DLOK_ERR**) because another connection has the record locked and the reader requested non-blocking locking.	The physical record read function returns error 42.
The record is committed (already, or when the lock is released).	The record read proceeds as usual, and the record that would have been skipped is returned to the caller.
The record is deleted or is a “resource.”	The physical record read function continues scanning the data file, reading the next record.
The record header contains an invalid record mark.	The physical record read function returns error 158 (**VFLG_ERR**).

Further changes have been introduced to reduce the occurrence of the 158 (**VFLG_ERR**) without skipping any record by changing the record header marker management during record addition.

`COMPATIBILITY NO_VFLG_ERR` can be used to disable this new handling and restore the earlier (post V8.14) behavior for the rare circumstance in which the old behavior is desired.

21.10 Linux File System Performance and Safety

Review the FairCom website for important information about c-treeACE performance and integrity on Linux platforms: **Linux Caching Considerations** (https://docs.faircom.com/doc**/ctserver/linux-caching-considerations.htm**).

21.11 COMMIT_DELAY Configuration Now Defaults to 1 ms on Linux Systems

The default FairCom Server configuration file, *ctsrvr.cfg*, included in c-treeACE Linux packages has been modified to improve performance. The `COMMIT_DELAY` setting now defaults to a value of 1 (ms). (*Previously, COMMIT_DELAY was disabled for Linux platforms, see note below.*) One ms has been shown to produce much better performance with *TRNLOG* files on Linux systems.

Note: `COMMIT_DELAY` had been disabled in an earlier correction for a Linux bug, which was preventing synchronous writes from being flushed to disk. That setting degraded *TRNLOG* performance under those circumstances and a default setting of 1 ms is again recommended with c-treeACE V10.3.1.21834 (Build 140307) or later.

21.12 Relaxed COMPATIBILITY FORCE_WRITETHRU Defaults

`COMPATIBILITY FORCE_WRITETHRU` enables the *WRITETHRU* filemode for all non-transaction files (which includes *PREIMG* files). The performance impact of *WRITETHRU* depends on whether `COMPATIBILITY PREV610A_FLUSH` is also enabled. See the descriptions in `COMPATIBILITY FORCE_WRITETHRU` and `COMPATIBILITY PREV610A_FLUSH`.

These options provide a good balance between update performance and recoverability of data in the event of an abnormal FairCom Server termination. They affect non-transaction files as follows:

- Non-transaction files that are neither created nor opened with the *WRITETHRU* filemode have their updates written to the FairCom Server cache and eventually written to the file system cache and to disk.
- Non-transaction files that are created or opened with the *WRITETHRU* filemode have their updates flushed to the file system cache.

Be sure to understand the impact of these file modes with respect to your file transaction mode in use, and type and vulnerability of your application data.

21.13 New Extended Data Types Support

The range of values available for new c-treeACE data types has been extended.

This support required changing the format in which the record schema (*DODA*) is stored on disk and in memory. It affects server, client, and standalone code.

Backward compatibility has been preserved as much as possible:

- The record schema is stored in the new format only if the record schema uses data types in the new range.
- A FairCom Server that supports the new range of data types sends the appropriate record schema format to the client based on whether the client supports the new range of data types. If the record schema uses a data type in the new range and the client does not support the new record schema format, the server fails the operation with error **DTPC_ERR** (1010, "This file uses data types that your client library does not support. Update your client library").
- A client that supports the new range of data types sends the appropriate record schema format to the server based on whether the server supports the new range of data types. If the record schema uses a data type in the new range and the server does not support the new record schema format, the client fails the operation with error **DTPS_ERR** (1011, "This client library uses data type support that your server does not support. Update your c-tree Server").
- A file whose record schema is stored in the new format has a bit set in the file's header to indicate that it uses the new format. A FairCom Server or standalone library that does not support extended data types will fail to open a file whose record schema is stored in the new format with error **FREL_ERR** (744, "file requires unavailable feature"), and a message is logged to *CTSTATUS.FCS* to indicate which DEF_MASK bit triggered the error. The bit that

indicates support for the new c-treeACE data type range is 0x40000, so the error message will look similar to the following (the message will include the 0x40000 bit in addition to any other feature bits that the server does not support):

```
FREL_ERR(744) unsupported DEF_MASK bits: 00040000x for file...
```

21.14 Dynamic Dump Stream Files No Longer Segment by Default

Nearly all of today's modern filesystems support file sizes larger than 2GB. Dump streams from the c-treeACE Dynamic Dump backup are frequently much larger than 2GB and, by default, are written into 2GB segments. This is proving to make management of the backup process more cumbersome. As a result, the Dynamic Dump backup option `EXT_SIZE` script option now defaults to `NO`, such that the dump backup is written to a single file. Simply specify an extent size in your dump script files if you wish to again segment your files.

21.15 Auto-Numbering Replication Defaults Changed

In this release, changes were made to the FairCom Server defaults for replication behavior on *SRLSEG* and identity fields.

Compatibility Note: This is a change in behavior, but many of the FairCom Server configuration files included in FairCom products already contain this option so the behavior is already in effect in those cases. However, the FairCom RTG package did not include these options, so this will change the default behavior for FairCom RTG COBOL and FairCom RTG BTRV.

The default values have been changed for the following options:

REPL_SRLSEG_ALLOW_UNQKEY now defaults to YES

This option allows a *SRLSEG* index to be used as a replication unique index. Without this option, a data file whose only unique index is a *SRLSEG* index would not qualify for replication. By changing this default to YES, the *SRLSEG* index can be used as a replication unique index, which is the commonly-expected behavior.

REPL_SRLSEG_USE_SOURCE now defaults to YES

This option uses the serial number value from the source FairCom Server when adding a record to a replicated file. This option applies to c-tree's asynchronous (source/target) replication model, in which a Replication Agent replicates changes from a source FairCom Server to a target FairCom Server. By changing this default to YES, records added to the target server will contain the *SRLSEG* value from the record on the source server, rather than the target server generating its own *SRLSEG* value for the new record. This default option corresponds to the behavior that is most likely to be expected.

REPL_SRLSEG_USE_MASTER now defaults to YES

This option uses the serial number value from the master FairCom Server when adding a record to a replicated file. This option applies to c-tree's synchronous (local/master) replication model, in which an update to a record on a local FairCom Server triggers an update to the record in the associated table on the master FairCom Server. By changing this default to YES, records added to the local server will contain the *SRLSEG* value from the record on the master server, rather than the local server generating its own SRLSEG value for the new record. This default option corresponds to the behavior that is most likely to be expected.

REPL_IDENTITY_USE_SOURCE now defaults to YES

This option uses the identity field value from the source FairCom Server when adding a record to a replicated file. This option applies to c-tree's asynchronous (source/target) replication model, in which a Replication Agent replicates changes from a source FairCom Server to a target FairCom Server. By changing this default to YES, records added to the target server will contain the identity field value from the record on the source server, rather than the target server generating its own identity field value for the new record. This default option corresponds to the behavior that is most likely to be expected.

REPL_IDENTITY_USE_MASTER now defaults to YES

This option uses the identity field value from the master FairCom Server when adding a record to a replicated file. This option applies to c-tree's synchronous (local/master) replication model, in which an update to a record on a local FairCom Server triggers an update to the record in the associated table on the master FairCom Server. By changing this default to YES, records added to the local server will contain the identity field value from the record on the master server, rather than the local server generating its own identity field value for the new record. This default option corresponds to the behavior that is most likely to be expected.

21.16 “Add Unique Keys First” Feature Applied to ctADD2END Files

Variable-length data files using our *ctADD2END* feature now automatically enable a "add unique keys first" optimization. Enabling this optimization prevents space from being allocated in the data file that is not reused if a record ADD operation fails with a duplicate key error.

21.17 Maximum LIST_MEMORY Setting Increased to 10 MB

The maximum value for the `LIST_MEMORY` server administrator keyword has been changed from 1 MB to 10 MB.

21.18 Maximum Index Members per File (MAXMEMB)

The default for the `MAXMEMB` (the number of index members per single index file) has been updated to 127 allowing a larger number of segments per index.

We now define ctMAXMEMB in *ctopt1.h* instead of in *ctopt2.h*, because *ctport.h* references ctMAXMEMB and it is included before *ctopt2.h*.

We also define MAXMEMB to ctMAXMEMB in *ctopt1.h*.

21.19 Maximum Number of Indexes per Data File (MAX_DAT_KEY) Default Increased to 64

Occasionally data files require a large number of indexes. Commencing with c-treeACE V10.3, the number of indexes default limit was increased from 32 to 64. Customers using the FairCom Server can use the `MAX_DAT_KEY` keyword to change the limit on the number of indexes per data file.

If this limit is too low, the typical error code that would be seen is error 107, **IDRK_ERR** "too many keys for ISAM data file."

This value affects the amount of memory that is allocated to store ISAM index state information. It can be increased without a major impact on performance unless FairCom Server is being run in an environment with very little memory (e.g., certain embedded applications).

Note: In the *standalone* model, `MAX_DAT_KEY` is a *compile-time* setting. If this value is changed, the code must be recompiled with the new setting.

21.20 Maximum Number of Open Files per User (MAX_FILES_PER_USER) Default Increased to 32767

The default FairCom Server `MAX_FILES_PER_USER` value has been increased from 2048 to 32767. We increased the default value to avoid file open operations failing with error **46** due to the lower limit. Generally, developers expect a single connection can open as many files as FairCom Server's `FILES` limit, but this was not the case due to the previous per user default limit.

21.21 Allow a Single Byte or SByte to be passed as a BINARY Value in ADO.NET

The logic has been updated to allow a single Byte or SByte to be passed as a BINARY value in addition to Byte[]. Other types now throw an exception.

21.22 c-treeACE Memory Allocation Limit Disabled

Prior to this modification, FairCom Server's `LMT_MEMORY` configuration option defaulted to 128MB, meaning that a single memory allocation could not exceed 128MB. However, this limit restricted the size of variable-length records from FairCom Server and limited the amount of memory that could be allocated with the `RECOVER_MEMLOG` recovery option.

This limit has been *disabled* by default. If `LMT_MEMORY` is not specified, FairCom Server does not place a limit on the size of a single memory allocation. If desired, `LMT_MEMORY` can be specified in *ctsrvr.cfg* setting a desired maximum allocation size.

Note: This modification results in a behavior change.

This change also applies to standalone mode: `#define ctMEMLMT` is set to zero by default, meaning no memory allocation size limit. If desired, c-treeACE can be compiled with *ctMEMLMT* defined to a non-zero value.

21.23 c-treeACE SQL SETENV limit raised to 8192

c-treeACE SQL server did not start when *ctsrvr.cfg* contained a `SETENV` configuration referencing an environment variable evaluating to more than 1024 bytes.

Previously two limits were enforced in setting environment variables in *ctsrvr.cfg*:

1. While parsing *ctsrvr.cfg* - Limit of 1024 bytes to hold the content of *ctsrvr.cfg* and expand the environment variables referenced (an error was returned if the expanded value did not fit)
2. When storing the variable name and content for SQL usage - Limit of 50 bytes for the environment variable name and 255 bytes for the content (name/value truncated if it did not fit)

Limit #1 has now been increased to 8192

Limit #2 has been removed with dynamic memory allocation.

The net effect is a practical limit is 8192 bytes.

21.24 c-treeACE SQL Stored Procedure Server-side Debugging Options

In V11 and later, server-side stored procedure execution logging can be turned on and off using the TPESQLDBG environment variable. TPESQLDBG is an array of 'Y'/'N' characters that determine which debug options are enabled.

Prior to this revision, this was achieved by setting the 11th element (offset 10, `java_debug`) in the TPESQLDBG environment variable. The logging produced in this way is quite verbose and included information that was useful only to FairCom Technicians.

The interpretation of TPESQLDBG (and other methods setting the same debug features) has been modified as follows:

- TPESQLDBG (which was an array of 12 'Y'/'N' characters) has been expanded to 13 elements.
- The element at offset 12 is a new element named `stp_logging` that controls the **log()** method used in stored procedures to generate log messages.

Note that activating element #11 (`java_debug`) activates internal logging and **log()** method for ANY stored procedure language, not just Java.

21.25 Java Stored Procedure Runtime Classes no Longer Require ctreedbs in Path

Java stored procedure runtime support on the server has been enhanced such that, for ease of compilation and execution, it is no longer necessary to have the *ctreedbs* shared library/DLL in the path for libraries.

21.26 PHP - Components Now Match non-Thread-Safe Defaults for Windows IIS PHP Installations

The default PHP installation for Microsoft Windows IIS is not thread-safe, which implies that any PHP extensions should also be compiled as "not thread-safe." Previously, c-treeACE PHP components were compiled and distributed as thread-safe, which were incompatible with default PHP IIS installations. For better default compatibility, c-treeACE SQL PHP components are now compiled and distributed as non-thread-safe.

21.27 c-treeACE SQL JDBC Socket Timeout Defaults to 0

The default c-treeACE SQL JDBC socket timeout has been changed from 30 minutes to 0 (no timeout). This timeout can be modified by setting the `sock_timeout` property explicitly at connection time:

```
Properties info = new Properties();
info.setProperty("user","ADMIN");
info.setProperty("password","ADMIN");
info.setProperty("sock_timeout","30000");
conn = DriverManager.getConnection ("jdbc:ctree:6597@localhost:ctreeSQL", info);
```

21.28 c-treeACE JDBC Java 1.5 Compatible Driver Availability

c-treeACE JDBC requires JRE 1.6 or later. A Java 1.5 compatible version of the JDBC driver is available upon request for applications still tied to this legacy Java framework.

21.29 Windows Servers Now Statically Linked with ZLIB Compression Libraries

c-treeACE uses the open source ZLIB library as a data compression option. These were previously picked up as a shared library. ZLIB libraries are now statically linked into Windows build of c-treeACE servers intended for general distribution.

Note: Some customer OEM agreements prevent us from distributing open source code. In these cases provisions are made to not statically link with ZLIB and a compatible ZLIB .DLL will need to be provided on the environment where compression is desired.

22. Document History

12/02/2015	Updated *Millisecond Timestamp Resolution Support* (page 125) to indicate this feature is enabled in all V11 releases. Added related FairCom DB API functions.
11/24/2015	Added *Support creating advanced encryption master key store encrypted at system level on Windows* (page 240).
11/11/2015	First published.

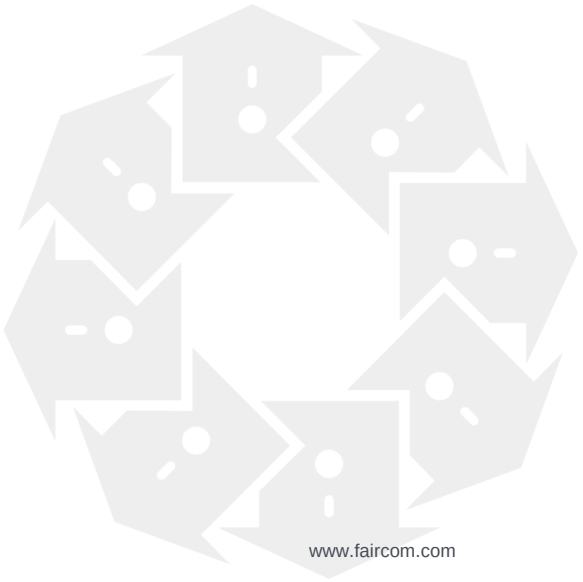
Last published Thursday, January 30, 2025.

23. 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

24. Index

D

E

F

U

V

W