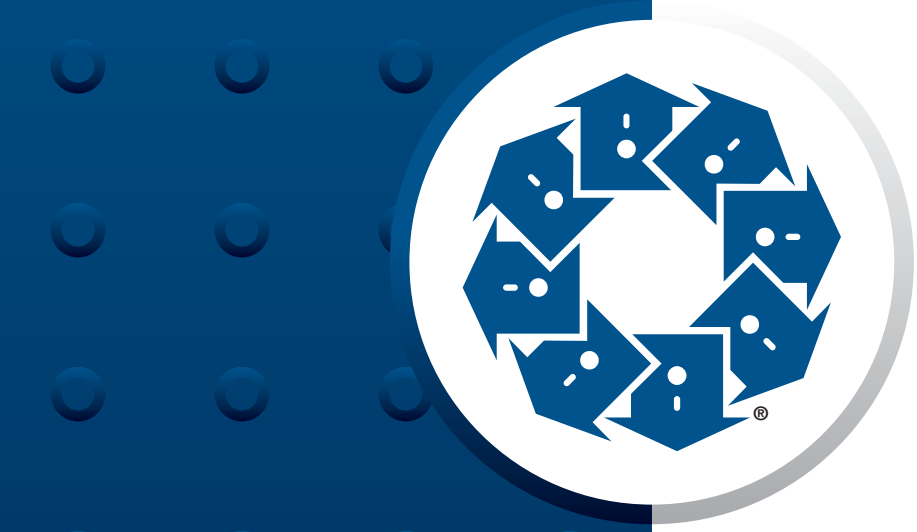


FairCom®

c-tree Plus®

V9

c-tree Server

Reference Guide

c-tree Server

Reference Guide

Information in this document is subject to change without notice.

Trademarks

FairCom welcomes your comments on this document and the software it describes. Send comments to:

Documentation Comments
FairCom Corporation
6300 W. Sugar Creek Drive
Columbia, MO 65203

6/26/2008

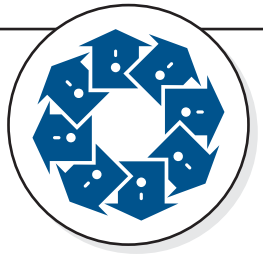
CONTENTS

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-tree Function name.
Parameter	c-tree Function Parameter.
Code Examples	Code example or Command line usage.
utility	c-tree executable or utility.
filename	c-tree file or path name.
CONFIGURATION KEYWORDS	c-treeACE Configuration Keyword.
BIG_ERR	c-tree Error Code.



System Architecture

This chapter discusses design considerations for systems that rely upon the c-tree Server. The first section presents an overview of ways to integrate the c-tree Server into a system's architecture. The second section discusses supported c-tree Server features and their effect on the availability of the system in the case of a catastrophic failure.

The c-tree Server is a multithreaded database server that provides client applications access to c-tree Plus data and index files. The server coordinates access by both local and remote clients to the data. Clients communicate with the server using the server's supported client application programming interfaces (APIs), which include the following:

- c-tree Plus lowlevel API
- c-tree Plus ISAM API
- c-treeDB API
- c-treeSQL API

The c-tree Server supports a variety of features including ISAM indexing using B+ trees, multiple levels of transaction control, dynamic backups, and custom server-side processing.

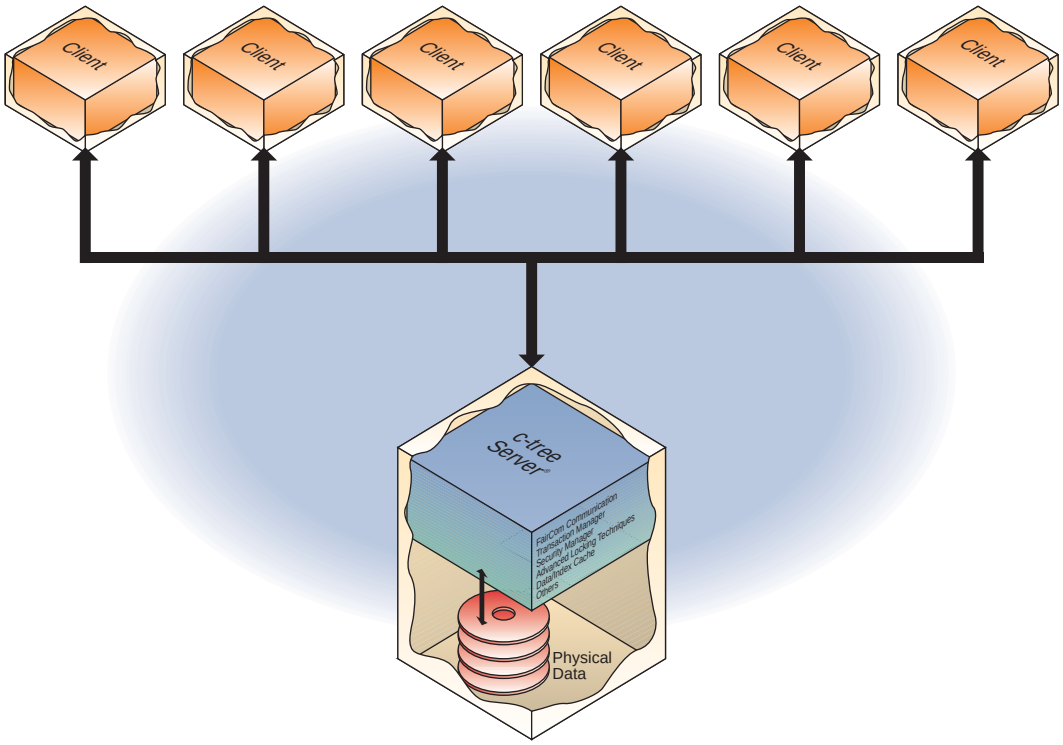
1.1 Positioning the c-tree Server in the System Architecture

When designing a system, the system architect determines the appropriate positioning of the c-tree Server in the system architecture based upon system requirements. In some cases simplicity is the primary requirement. In other cases, scalability and fault tolerance are the primary requirements. This section discusses two options for integrating the c-tree Server into a system architecture: the single c-tree Server architectural model and the multiple c-tree Server architectural model.

Single c-tree Server Architecture

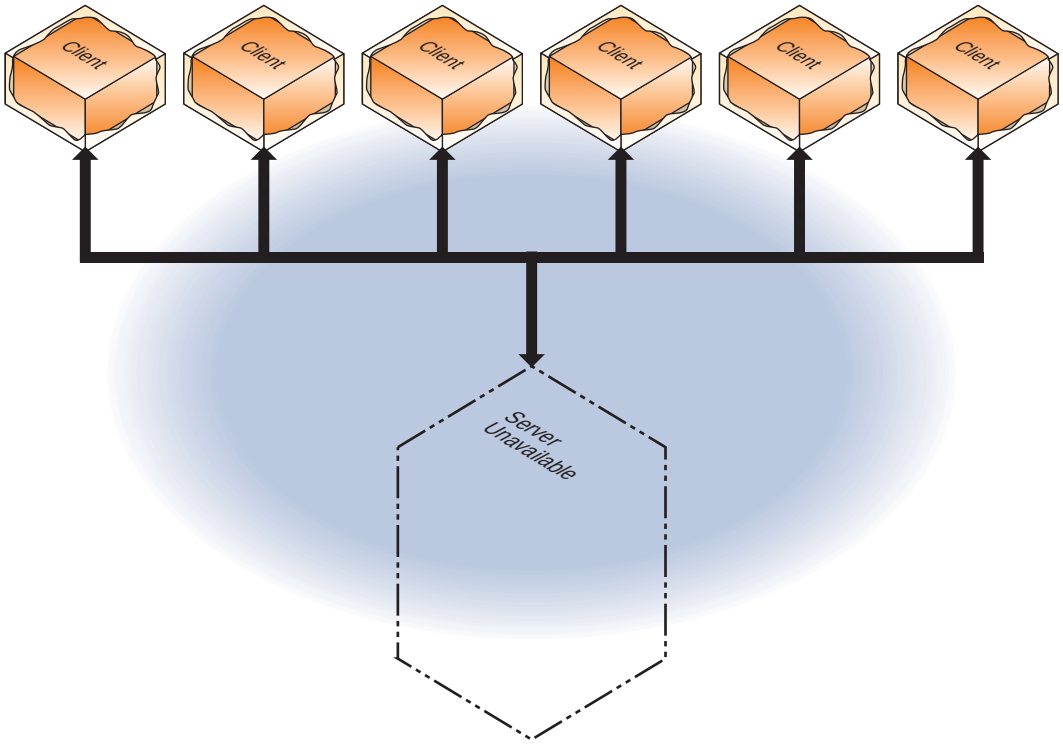
The single c-tree Server architectural model is the simpler of the two models. In this model, one c-tree Server serves all clients. The advantage of this model is its simplicity. Because this model involves only one c-tree Server, there is no application routing logic and no synchronization of data among multiple servers. The application simply connects to the server and accesses the data.

Figure 1: Single c-tree Server Architecture



The tradeoff for the simplicity of this approach is that the client load and scalability of the database system is limited to the capacity of the machine on which the c-tree Server runs. This choice of system architecture limits the ability of the system to scale to meet increased capacity requirements over time. Also, the availability of the system is determined by the availability of the single c-tree Server process and the machine on which it runs. If a software or hardware component failure renders the c-tree Server process or its data inaccessible, the availability of the system is directly affected.

Figure 2: Effect of Unavailable Server in Single c-tree Server Architecture



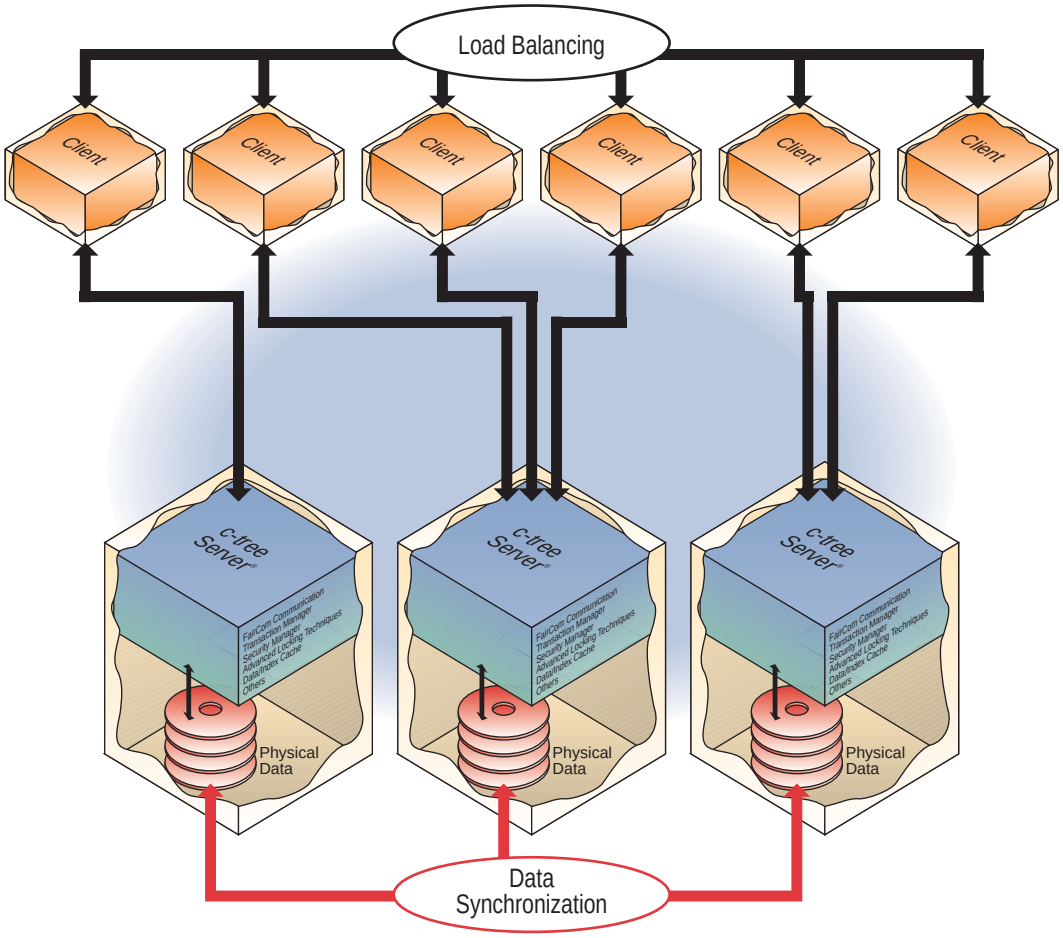
Multiple c-tree Server Architecture

Given the scalability and fault tolerance implications of a single c-tree Server model, many enterprise-level systems choose to implement a multiple c-tree Server architectural model. In this model, multiple c-tree Servers serve clients, and the architecture frequently includes load balancing and data synchronization components.

Load balancing components route incoming requests to c-tree Servers in such a way as to spread the load evenly among the servers. The use of load balancing enables efficient use of multiple systems, enhancing scalability of the system.

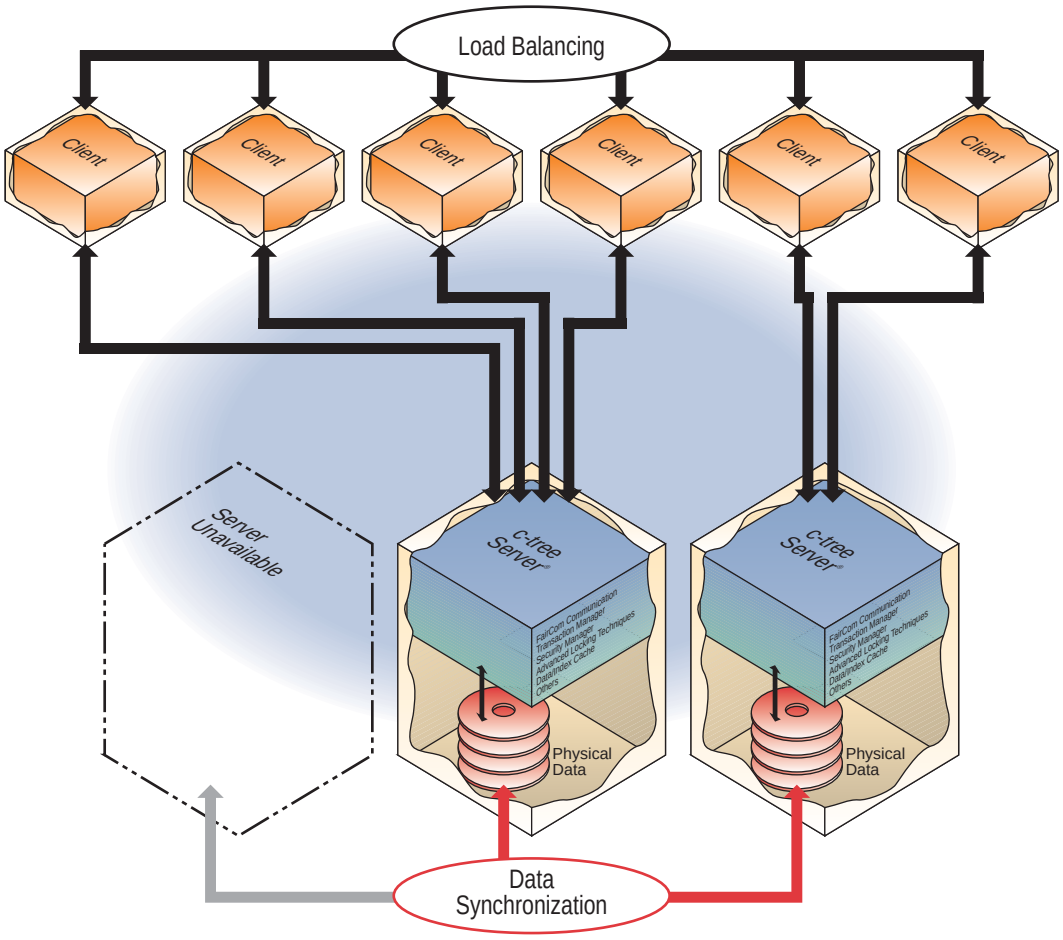
In a system with more than one c-tree Server, each c-tree Server manages its own data set. The system architect decides how to partition data sets among c tree Servers. For example, the design could specify that each server maintains a full copy of the data set, or that each server maintains a subset of the data set. Data synchronization components apply changes made to one data set to other data sets as required by the system.

Figure 3: Multiple c-tree Server Architecture



In the event of a failure in one portion of the system, client access is automatically rerouted ensuring continuous operations.

Figure 4: Effect of Unavailable Server in Multiple c-tree Server Architecture

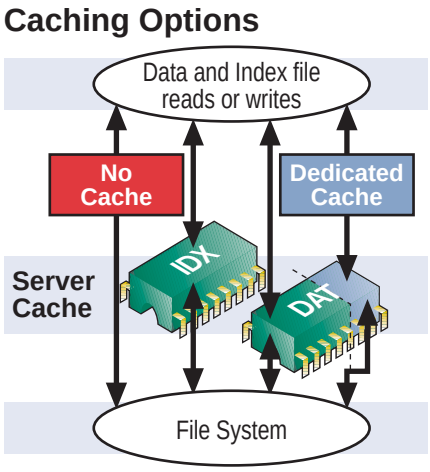


1.2 Selecting c-tree Server Features Used in the System

The c-tree Server supports many options that affect the availability of the server's data over time and the state of the data in the event of a catastrophic failure. Many of the features discussed in this section can be configured on a file-by-file basis. The system architects and application designers can determine the appropriate features to use for each data and index file based on the role the files play in the system and by understanding the effect of these features on availability and recovery of the data.

Data and Index Caching Options

Figure 5: c-tree Server Caching Options



The c-tree Server maintains data and index caches in which it stores the most recently used data images and index nodes. The caches provide high-speed memory access to data record images and index nodes, reducing demand for (slower) disk I/O on data and index files. The server writes data and index updates first to cache and eventually to disk. Data and index reads are satisfied from the server's cache if a cache page contains the requested data record image or index node. When necessary, the server writes pages to disk using a least recently used (LRU) scheme. The server also supports background flushing of cache buffers during system idle times.

The size of the data and index caches are determined by server configuration file settings. It is also possible to disable caching of specific data files and to dedicate a portion of the data cache to specific data files.

Caching of Data Records

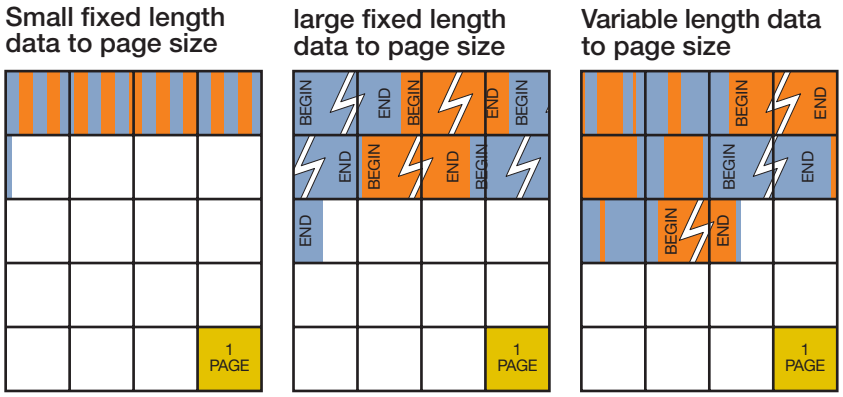
The c-tree data cache uses the following approach to cache data record images: if the data record fits entirely within one or two cache pages, then the entire record is stored in the cache.

Note: Even a record smaller than a single cache page may require two cache pages as the record position is generally not aligned on a cache page boundary.

If the data record image covers more than two cache pages, then the first and last segments of the record are store in the cache, but the middle portion is read from or written to the disk. These direct I/O's are generally efficient operations since they are aligned and are for a multiple of the cache page size.

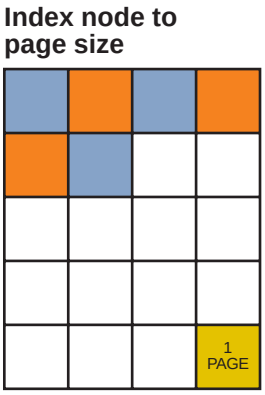
The nature of this approach can be modified. If the server keyword `MPAGE_CACHE` is set to a value greater than zero, say N, then records that fall within N+2 cache pages will be stored entirely within the cache. The default value is zero. This causes the cache to behave as described above.

Note: Setting `MPAGE_CACHE` greater than zero does NOT ensure faster system operation. It is more likely to be slower than faster. It does cause more of a record to be in cache, but there is increased overhead managing each individual cache page. The cache pages for consecutive segments of a record (where a segment fills a cache page) are completely independent of each other. They are not stored in consecutive memory. And I/O is performed separately for each cache page. This configuration option should only be used for special circumstances with careful, realistic testing.



Caching of Index Nodes

Figure 6: Data Length to Page Size



To understand the c-tree Server's caching of index nodes, it is necessary to review the relationship between the server's *PAGE_SIZE* setting, the size of index cache pages, and the size of index nodes.

The server's *PAGE_SIZE* setting determines the size of index cache pages and the size of index nodes for index files created by the server. Because the server must be able to fit an index node into a single index cache page, the server's *PAGE_SIZE* setting determines the maximum size of index nodes supported by the server. The server can access index files that were created with an index node size less than or equal to the server's *PAGE_SIZE* setting, but attempting to open an index whose index node size exceeds the server's *PAGE_SIZE* fails with error **KSIZ_ERR** (40, index node size too large).

Properties of Cached Files

Although caching data benefits server performance, it is important to be aware of the effect of caching data on the recoverability of updates. The state of a cached file after an abnormal server termination depends on the c-tree options in effect for that file. Below is a summary of the effect of caching on each file type. The following sections discuss this topic in detail.

- *TRNLOG* files: Caching does not affect recoverability. The server's transaction processing logic ensures that all committed transactions are recovered in the event of an abnormal server termination.
- *PREIMG* or non-transaction files: Caching can lead to loss of unwritten cached data in the event of an abnormal server termination. For these file types, the server does not guarantee a persistent version of the unwritten updated cache images exists on disk, so any unwritten cached data is lost in the event of an abnormal server termination.

- *WRITETHRU* (applies to *PREIMG* and non-transaction files): To minimize loss of cached data, the *WRITETHRU* attribute can be applied to a *PREIMG* or non-transaction file. *WRITETHRU* causes writes to be written through the server's cache to the filesystem (for low-level updates) or flushed to disk (for ISAM updates).

Configuring Caching

The memory allocated to the data and index caches is set in the server configuration file (ctsrvr.cfg) using the `DAT_MEMORY` and `IDX_MEMORY` options. For example, to allocate 1 GB data cache and 1 GB index cache, specify these settings in *ctsrvr.cfg*:

```
DAT_MEMORY 1073741824
IDX_MEMORY 1073741824
```

The server allocates this memory at startup and partitions it into cache pages. The server's *PAGE_SIZE* setting determines the size of the data and index cache pages.

The maximum size of data and index caches is limited to 2 GB unless the following server configuration keyword is used:

```
COMPATIBILITY LARGE_CACHE
```

Adding `COMPATIBILITY LARGE_CACHE` to the configuration file permits the `DAT_MEMORY` and `IDX_MEMORY` values to be reinterpreted as megabytes instead of bytes. If the byte value in the configuration file is less than or equal to 64000, then the value is reinterpreted as megabytes. This permits up to 64GB of index or data cache to be requested. If the value is greater than 64000, it is interpreted as bytes (just as without the `LARGE_CACHE` option). If the `LARGE_CACHE` option is not used, the values for `DAT_MEMORY` and `IDX_MEMORY` are interpreted as bytes, regardless of their values. This option is only available on systems that support 8-byte integers.

In some cases it may be advantageous to turn off data file caching, especially for files with very long variable length records. For a server, configuration entries of the form:

```
NO_CACHE <data file name>
```

cause caching for the specified data files to be disabled. Note that *<data file name>* may specify a wildcard pattern.

Cache memory can be dedicated to specific data files using the following server configuration option:

```
SPECIAL_CACHE_FILE <data file name>#<bytes dedicated>
```

By default, the c-tree Server creates two idle flush threads at startup. One thread flushes transaction file buffers, and the other flushes non-transaction file buffers. The threads wake up periodically and check if the server is idle. If so, the flush is begun. If subsequent server activity causes the server to no longer be idle, then the flushes are terminated. Low priority background threads, such as the delete node thread, do not affect the server's idle state. c-tree clients and c-treeSQL clients and checkpoints do cause the idle status to be modified.

The configuration file can modify the characteristics of these idle flush threads:

```
IDLE_TRANFLUSH <idle check-interval for tran files>
IDLE_NONTRANFLUSH<idle check-interval for nontran files>
```

The idle check interval values are specified in seconds. Setting the interval to zero or a negative value disables the thread. The defaults for these intervals are each set at 15 seconds.

Transaction-Controlled Files

An application can use the c-tree Server's transaction capabilities to guarantee atomicity and, optionally, recoverability of data. The c-tree Server supports the following transaction processing options:

- *PREIMG* (provides atomicity without recoverability)
- *TRNLOG* (provides atomicity with recoverability)

- *TRNLOG* with *LOGIDX* indexes (provides atomicity with recoverability, with logging of index reorganization operations to speed automatic recovery).

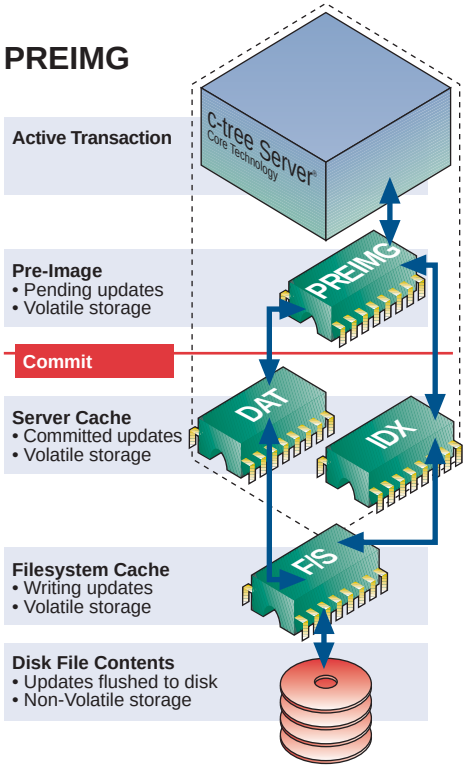
The application selects the transaction processing options that are in effect for the file when it creates data and index files using c-tree Plus file creation API functions. The following sections discuss the properties of each transaction processing option, including their effect on the state of files in the event of an abnormal server termination.

PREIMG Transaction Files

The *PREIMG* (“pre-image”) transaction mode supports transaction atomicity but not transaction recoverability. *PREIMG* files may be updated only within an active transaction. The server stores *PREIMG* file updates in memory known as ‘preimage space’ until the transaction is committed or aborted. This use of preimage space guarantees atomicity and isolation of transaction operations: changes are made on an all or nothing basis and other clients do not see changes until the transaction is committed.

Properties of PREIMG Files

Figure 7: c-tree Server PreImage File Handling



The state of a *PREIMG* file at a point in time is stored in a combination of the following locations:

- Updated buffers held in preimage space. (Pending updates held in server memory: this is volatile storage)
- Updated server data/index cache pages. (Committed updates held in server memory: this is volatile storage)
- Filesystem cache pages. (Committed updates written to filesystem but not flushed to disk held in system memory: this is volatile storage)

- Disk file contents. (Committed updates written to filesystem and flushed to disk: this is non-volatile storage)

Only the disk file contents are persistent and unlike *TRNLOG* transaction file updates (discussed below), *PREIMG* file updates are not logged to the server's transaction logs. For this reason, *PREIMG* files are not recoverable in the event of an abnormal server termination. In such a situation a *PREIMG* file is in an unknown state because an unknown number of updates may have not yet been written to disk at the time of the abnormal server termination. Also, because automatic recovery does not process *PREIMG* files, a *PREIMG* file rebuilt after an abnormal server termination is not guaranteed to be in a consistent transaction state. In such a situation, *PREIMG* files could contain data that was in the process of being committed but for which the commit had not yet been completed.

Backup and Restore Options for PREIMG Files

Backup copies of *PREIMG* files can be made using the following approaches:

Online backup using dynamic dump

Although automatic recovery is not available to *PREIMG* files, it is possible to perform periodic dynamic backups of *PREIMG* files using the c-tree Server's dynamic dump feature. Using the PREIMAGE_DUMP dynamic dump script option promotes *PREIMG* files to full *TRNLOG* files during the dynamic dump process. The promotion to a *TRNLOG* file means that a full transaction log is maintained during the dump process. This process guarantees that any changes made to the files during the backup are saved in these specially maintained transaction logs. The ability to dynamically back up user data files minimizes the loss of the automatic recovery feature with this mode. The dynamic dump produces a dump stream file containing the disk contents of the *PREIMG* files and the changes made during the dynamic dump. If it becomes necessary to restore the backup copy of the *PREIMG* files, the **ctrdmp** utility can be used to extract the *PREIMG* files from the dump stream file, restoring them to their state at the time of the backup.

Offline backup using file copy

An offline backup of *PREIMG* files can be performed using system file copy utilities when the server is shut down or when the server is running and the files to be backed up are closed. It is important to ensure that the server is shut down or the files are closed before making a backup copy of files using a system file copy utility in order to ensure that all updated buffers are flushed to disk. Otherwise, data may be lost and the file may be in an inconsistent state.

When to Use PREIMG Files

The benefit of *PREIMG* is that it avoids the overhead associated with writing to and flushing the transaction logs. If atomicity is required for a file but recoverability is not, *PREIMG* may be an appropriate choice. Some specific cases in which *PREIMG* may be appropriate include:

- Using *TRNLOG* data files and *PREIMG* indexes if willing to rebuild the indexes after an abnormal server termination.
- Using *PREIMG* on files that can be re-created in the event of an abnormal server termination.

To minimize loss of unwritten cached *PREIMG* file updates in the event of an abnormal server termination, consider using *WRITETHRU* for *PREIMG* files or periodically calling the c tree API function **CtreeFlushFile()** to flush *PREIMG* data and index cache pages to disk.

Creating and Using PREIMG Files

To create a *PREIMG* data or index file, include the *PREIMG* filemode in the filemode value specified when creating the file.

- If using an ISAM-level file creation function, include *PREIMG* in the *dfilmod* and *ifilmod* fields of the *IFIL* structure supplied to the function.
- If using a low-level file creation function, include *PREIMG* in the filemode parameter supplied to the function.

The c-tree Server allows updates to *PREIMG* files only within an active transaction. To start a transaction, call the c-tree Plus API function **Begin()** with a transaction mode of either *PREIMG* or *TRNLOG*. A *PREIMG* file may participate in either a *PREIMG* or a *TRNLOG* transaction. Regardless of which transaction type is specified, updates to *PREIMG* files are not logged to the server's transaction logs. An update on a *PREIMG* file that is attempted outside a transaction fails with c-tree error **TTYP_ERR** (99, transaction type/filmod conflict).

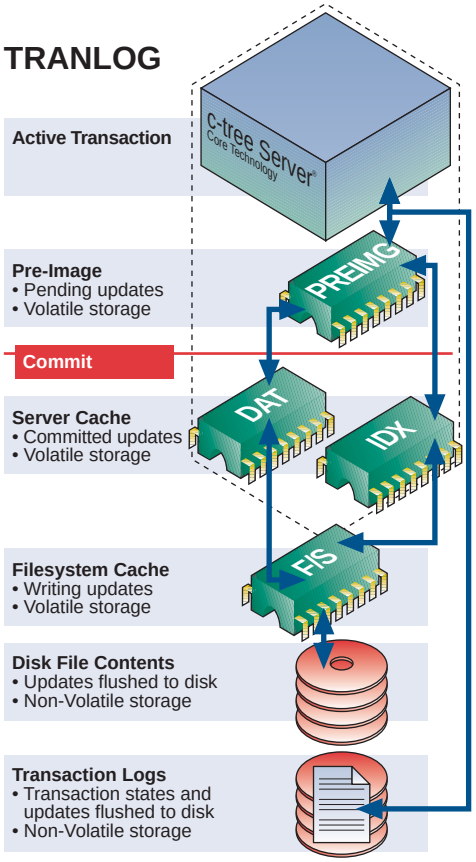
TRNLOG Transaction Files

The *TRNLOG* ("tran-log") transaction mode supports both transaction atomicity and transaction recoverability. *TRNLOG* files may be updated only within an active transaction. The server stores *TRNLOG* file updates in memory known as "preimage space" until the transaction is committed or aborted and logs transaction begin and commit operations and file updates to disk-based transaction log files. The use of preimage space guarantees atomicity and isolation of transaction operations: changes are made on an all or nothing basis and other clients do not see changes until the transaction is committed. The use of transaction logs guarantees recoverability of transactions in the event of an abnormal server termination.

Properties of TRNLOG Files

Because the c-tree Server logs updates made to *TRNLOG* files to its transaction logs, *TRNLOG* files are fully recoverable in the event of an abnormal server termination. The server ensures *TRNLOG* files are in a consistent state by performing automatic recovery at server startup. The server's automatic recovery procedure involves scanning the transaction log to determine which transactions must be undone and which must be redone, based on the transaction log entries and the state of the files involved in the transactions.

Figure 8: c-tree Server TRANLOG File Handling



The state of a *TRNLOG* file at a point in time is stored in a combination of the following locations:

- Updated buffers held in preimage space. (Pending updates held in server memory: this is volatile storage)
- Updated server data/index cache pages. (Committed updates held in server memory: this is volatile storage)
- Filesystem cache pages. (Committed updates written to filesystem but not flushed to disk held in system memory: this is volatile storage)
- Disk file contents. (Committed updates written to filesystem and flushed to disk: this is non-volatile storage)
- Transaction log contents. (Transaction state entries and file updates flushed to disk: this is non-volatile storage)

Only the disk file and transaction log contents are persistent. The c-tree Server can guarantee recoverability of *TRNLOG* files in the event of an abnormal server termination because it logs to disk the

transaction state and updated buffers necessary to guarantee recoverability. At startup, the automatic recovery procedure applies the necessary changes to *TRNLOG* data and index files to ensure the system is in a consistent transaction state.

Backup and Restore Options for TRNLOG Files

Backup copies of *TRNLOG* files can be made using the following approaches:

Online backup using dynamic dump

The c-tree Server's dynamic dump feature can be used to make a consistent point in time backup of *TRNLOG* files. The server maintains a full transaction log during the dump process and produces a dump stream file containing the disk contents of the *TRNLOG* files and the changes made during the dynamic dump. If it becomes necessary to restore the backup copy of the *TRNLOG* files, the **ctrdmp** utility can be used to extract the *TRNLOG* files from the dump stream file, restoring them to their state at the time of the backup.

Online backup using system disk snapshot

Some systems provide the ability to perform an instantaneous transparent copy of the contents of a disk. This approach can be used to backup *TRNLOG* files provided that the copy operation produces a point in time image of the disk and the server's transaction logs are included with the *TRNLOG* files. If this is done, the *TRNLOG* files can be restored to a consistent transaction state corresponding to the time the backup was taken by restoring the saved files (transaction logs and *TRNLOG* files), then starting the server and allowing it to perform automatic recovery on the *TRNLOG* files.

Offline backup using file copy

An offline backup of *TRNLOG* files can be performed using system file copy utilities when the server is shut down or when the server is running and the files to be backed up are closed. It is important to ensure that the server is shut down or the files are closed before making a backup copy of files using a system file copy utility in order to ensure that all updated buffers are flushed to disk. Otherwise, data may be lost and the file may be in an inconsistent state.

When to Use TRNLOG Files

Create data and index files as *TRNLOG* files when operations on the files must be atomic and updates must be recoverable in the event of an abnormal server termination. If only atomicity is needed, *PREIMG* may be more appropriate.

The performance impact of *TRNLOG* due to checkpoint operations, transaction log flushing and transaction file buffer flushing can be minimized using transaction-related server configuration options such as:

```
CHECKPOINT_FLUSH
CHECKPOINT_INTERVAL
COMMIT_DELAY
LOG_SPACE
LOG_TEMPLATE
TRANSACTION_FLUSH
```

Creating and Using TRNLOG Files

To create a *TRNLOG* data or index file, include the *TRNLOG* filemode in the *filemode* value specified when creating the file.

- If using an ISAM-level file creation function, include *TRNLOG* in the *dfilmod* and *ifilmod* fields of the IFIL structure supplied to the function.

- If using a low-level file creation function, include *TRNLOG* in the filemode parameter supplied to the function.

The c-tree Server allows updates to *TRNLOG* files only within an active *TRNLOG* transaction. To start a *TRNLOG* transaction, call the c-tree Plus API function **Begin()** with a transaction mode of *TRNLOG*. An update on a *TRNLOG* file that is attempted outside a *TRNLOG* transaction fails with c-tree error **TTYP_ERR** (99, transaction type/filmod conflict).

TRNLOG Transaction Files with LOGIDX Indexes

TRNLOG indexes can optionally be created with the *LOGIDX* attribute. This attribute causes the server to log index reorganization operations to the transaction logs in order to facilitate automatic recovery of large indexes.

During automatic recovery, the c-tree Server scans the transaction logs in order to determine which *TRNLOG* index files must be processed by automatic recovery. For *TRNLOG* index files that were not created with the *LOGIDX* attribute, the server scans the leaf nodes, verifying links and rebuilding the upper part of the index tree. For large indexes, scanning the leaf nodes can be time-consuming.

Creating *TRNLOG* indexes with the *LOGIDX* attribute allows the server to make local repairs to index nodes (made possible by the additional LOGIDX transaction log entries) rather than requiring a full index leaf node scan, verification, and reconstruction. For this reason, using *LOGIDX* indexes can significantly speed up recovery time in the case of large *TRNLOG* indexes.

Using the *LOGIDX* attribute for an index file causes the c-tree Server to perform the following additional steps for index reorganization operations on that index:

- Write begin/end entries to the transaction log for index reorganizations (i.e., node splits or underflow node recovery). Before the “end” entry is written to the log, the index updates are saved to disk. If a begin is found without a corresponding end during automatic recovery, then the index must be examined in the region of the reorganization.
- Write vulnerable node entries to the transaction log to identify nodes which must be cleaned during automatic recovery. It is not necessary to place images of nodes in the log. Instead, the server logs the node position and optionally a key value which identifies the node’s logical position in the index.

Properties of LOGIDX Index Files

The format of a *TRNLOG* index created with the *LOGIDX* attribute is identical to that of a *TRNLOG* index created without the *LOGIDX* attribute. The only difference is the generation of additional log entries for index reorganization operations, which automatic recovery uses to speed recovery of the indexes as described above.

When to Use LOGIDX Index Files

The *LOGIDX* attribute is appropriate to use when a database contains large *TRNLOG* indexes and the application requires fast automatic recovery. The tradeoff for faster automatic recovery is that using *LOGIDX* might introduce a slight performance penalty for index update operations due to the generation of additional log entries and forced flushing of updated index buffers.

Creating and Using LOGIDX Index Files

To create a *TRNLOG* index file with *LOGIDX* support enabled, include the *TRNLOG* and *LOGIDX* filemodes in the filemode value specified when creating the index file.

- If using an ISAM-level file creation function, include *TRNLOG* | *LOGIDX* in the *ifilmod* field of the IFIL structure supplied to the function.

- If using a low-level file creation function, include *TRNLOG* | *LOGIDX* in the *filemode* parameter supplied to the function.

Because LOGIDX affects only the server's transaction processing behavior and does not affect the format of the index file, *LOGIDX* can be enabled at run-time for all *TRNLOG* indexes by specifying the following server configuration option in the server configuration file, *ctsrvr.cfg*:

```
FORCE_LOGIDX YES
```

The `FORCE_LOGIDX` keyword provides a simple way to enable *LOGIDX* processing for *TRNLOG* indexes, including those that were not created with the *LOGIDX* attribute. When this option is specified in the server configuration file, all *TRNLOG* indexes are treated as *LOGIDX* files. Using this keyword is a good way to ensure that *LOGIDX* processing is in effect for all *TRNLOG* indexes in order to ensure fast recovery.

6-Byte Transaction Numbers

The c-tree Server associates a unique transaction number with every transaction. The transaction numbers assigned by the server start with transaction number 1 and are ever-increasing during the server's operation.

When a c-tree client begins a transaction, the return value is the transaction number assigned to that transaction (or 0 in the case of an error). The server writes transaction numbers to the following persistent locations:

- Transaction logs (the start files *S*.FCS*, and the log files *L*.FCS*)
- *TRNLOG* indexes (in leaf nodes and header)
- Superfile hosts (including the server's *FAIRCOM.FCS*)

Transaction Number Limitations Prior to V8

Versions 6 and 7 of the c-tree Server support a 4-byte transaction number, of which 30 bits form the transaction number. For this reason, the maximum transaction number these versions of the c-tree Server support is 1,073,741,808 (0x3ffffff0).

When the server detects that the current transaction number is approaching the transaction number limit, it logs messages to the server status log, *CTSTATUS.FCS*, to inform the server administrator of an impending transaction number overflow. When the server detects a transaction number overflow it shuts down.

In the event of an impending transaction number overflow, the server administrator can follow these steps to restart the transaction numbering:

- Perform a clean shutdown of the c-tree Server as indicated by the "Perform system checkpoint" in the server status log.
- Delete the transaction logs: files *S0000000.FCS*, *S0000001.FCS*, and *L*.FCS*.
- Use the **ctcIntrn** utility on all *TRNLOG* indexes and superfile hosts (this includes the c-tree Server files *FAIRCOM.FCS* and *SYSLOGIX.FCS*) to reset the transaction numbers in the leaf nodes and index header.
- Restart the c-tree Server. The server creates new transaction logs and starts numbering transactions from 1 again.

Note: When the c-tree Server opens a *TRNLOG* index, the server compares the transaction number high water mark in the index header to the current transaction number. If the value in the index header is larger than the current transaction number and the open occurs outside a transaction, the server sets the current transaction number to the transaction number high water mark value from the index header. This behavior is significant because it means that opening an index containing a reference to

a large transaction number can cause the transaction number to jump to a large value. This could happen if an index was omitted from the **ctcIntrn** processing or an index was restored from a backup taken before the clean operation.

c-tree Server V8 Transaction Number Enhancements

For systems that must sustain a high transaction rate, the transaction number limit of the version 6 and version 7 c-tree Server can have significant implications for the availability of the c-tree Server. As an example, a system with a sustained rate of 1000 transactions per second would exceed the 4-byte transaction number limit in about 12 days of continuous operation.

To overcome this limitation, FairCom added support for 6-byte transaction numbers (also known as extended transaction numbers) to version 8 of the c-tree Server. The use of 6-byte transaction numbers (46 bits of which are used for the transaction number) provides the server the ability to sustain a rate of 1000 transactions per second for over 2000 years (as compared to the 12 day limit when using 4-byte transaction numbers).

The V8 c-tree Server always stores transaction numbers in the transaction logs as 6-byte values and stores transaction numbers as either 4-byte or 6-byte numbers depending on the index creation options. All indexes that the server creates for its own internal use are created as 6-byte transaction number indexes, so the only possible source of 4-byte transaction numbers is an application that creates index files as 4-byte transaction number files. If the application creates all its indexes as 6-byte transaction number files, the server will not be subject to the 4-byte transaction number limit.

When to Use 6-Byte Transaction Number Files

Use 6-byte transaction number files for high transaction rate systems to avoid the 4-byte transaction number limit, because the server must be shut down and indexes and superfile hosts cleaned when the transaction number limit is reached.

Creating 6-Byte Transaction Number Files

To create 6-byte transaction number files (this applies to superfile hosts and indexes), create an array of *XCREblk* (extended create block) structures, one for each physical data and index file to be created. Include the *ct6BTRAN* attribute in the *x8mode* field of each extended create block structure corresponding to an index file or superfile host. Call an *Xtd8* create function such as **CreateIFileXtd8()**, passing the extended create block array to the function.

In order to ensure that only 6-byte transaction number files are opened by the server, specify the following keyword in the server configuration file:

```
COMPATIBILITY EXTENDED_TRAN_ONLY
```

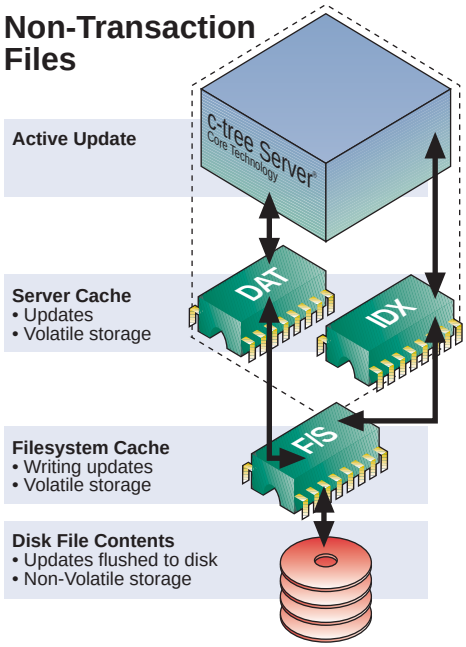
This keyword causes a **R6BT_ERR** (745, 6BTRAN file required) on an attempt to create or open a non-extended-transaction-number file. (Note that a read-only open is not a problem since the file cannot be updated.)

Non-Transaction Files

Depending on the application's requirements, it may be appropriate to create certain data and index files without transaction control. Non-transaction files avoid the overhead associated with transaction processing but do not guarantee atomicity of operations or recoverability of updates.

Properties of Non-Transaction Files

Figure 9: c-tree Server non-Transaction File Handling



The state of a non-transaction file at a point in time is stored in a combination of the following locations:

- Updated server data/index cache pages. (Updates held in server memory: this is volatile storage)
- Filesystem cache pages. (Updates written to filesystem but not flushed to disk held in system memory: this is volatile storage)
- Disk file contents. (Updates written to filesystem and flushed to disk: this is non-volatile storage)

The server does not guarantee that unwritten updated data and index cache pages are backed by a persistent copy on disk, so non-transaction files are not recoverable in the event of an abnormal server termination. In such a situation a non-transaction file is in an unknown state because an unknown number of updates may have not yet been written to disk at the time of the abnormal server termination.

Backup and Restore Options for Non-Transaction Files

Backup copies of non-transaction files can be made using the following approaches:

Online backup using dynamic dump

The c-tree Server's dynamic dump feature can be used to backup non-transaction files. However, unlike *PREIMG* and *TRNLOG* files the dynamic dump cannot guarantee a consistent point in time backup for non-transaction files. Non-transaction files are flushed if possible at the beginning of the dynamic dump. If successfully flushed and not updated during the dynamic dump, the file is marked clean in the dynamic dump stream file; otherwise it is marked dirty. At dump restore time, clean files have their update flags cleared but the update flag remains set for dirty files. A dirty restored file could be missing updates and dirty data and index files could be out of sync. Dirty restored files must be rebuilt to clear the update flag and to ensure consistency between data and index files.

Offline backup using file copy

An offline backup of *TRNLOG* files can be performed using system file copy utilities when the server is shut down or when the server is running and the files to be backed up are closed. It is important to

ensure that the server is shut down or the files are closed before making a backup copy of files using a system file copy utility in order to ensure that all updated buffers are flushed to disk. Otherwise, data may be lost and the file may be in an inconsistent state.

When to Use Non-Transaction Files

Use non-transaction files when the files are of a temporary nature and can be re-created or the data in the files can be restored from another source in the event of an abnormal server termination.

To minimize loss of unwritten cached non-transaction file updates in the event of an abnormal server termination, consider using *WRITETHRU* for non-transaction files or periodically calling the c tree API function **CtreeFlushFile()** to flush non-transaction data and index cache pages to disk.

Creating Non-Transaction Files

To create a non-transaction data or index file, do not include the *TRNLOG* or *PREIMG* file modes in the filemode value specified when creating the file.

- If using an ISAM-level file creation function, do not include *TRNLOG* or *PREIMG* in the *dfilmod* and *ifilmod* fields of the *IFIL* structure supplied to the function.
- If using a low-level file creation function, do not include *TRNLOG* or *PREIMG* in the filemode parameter supplied to the function.

WRITETHRU Files

For non-*WRITETHRU* files, the server stores data and index updates in its internal cache and does not write updates immediately to disk. For *TRNLOG* files, this is not a concern because committed updates to *TRNLOG* files are logged to the transaction logs. For non-*TRNLOG* files, however, in the event of an abnormal server termination the contents of the cache (and hence any unwritten updates to data and index files) will be lost.

In this situation, the server marks non-*TRNLOG* files as corrupt to indicate that the file was opened, updated and not properly closed, so its state is unknown. Attempting to open such a file fails with error **FCRP_ERR** (14, file corrupt at open). Rebuilding the data file and its associated indexes resets the update flag and allows the application to open the file, but all cached updates that had not yet been written to disk are lost.

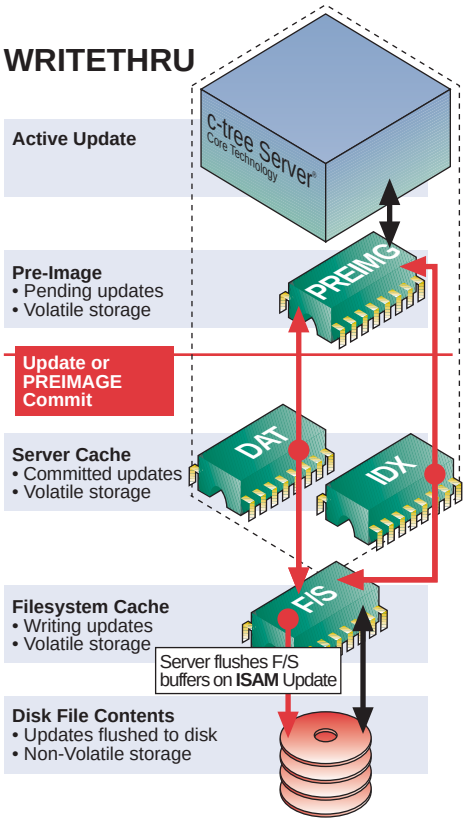
The *WRITETHRU* file mode can be applied to c-tree data and index files to cause the server to write updates through the server's cache to the filesystem cache or to disk, thereby minimizing the potential for loss of cached updates in the event of an abnormal server termination. While ensuring the updates are written to the filesystem or to disk, *WRITETHRU* preserves the updates in the server's cache so that reads can be satisfied from the server's cache.

A data or index file can be created as a *WRITETHRU* file (in which case *WRITETHRU* is a permanent attribute of the file), or it can be opened as a *WRITETHRU* file (in which case the file is treated as *WRITETHRU* until it is closed).

Properties of WRITETHRU Files

For non-transaction *WRITETHRU* files, all updates are written through the server's cache to the filesystem cache. In addition, the server flushes filesystem buffers on each ISAM update operation for *WRITETHRU* files. (Low-level updates on *WRITETHRU* files are written through the server's cache to the filesystem cache but are not flushed to disk.)

Figure 10: c-tree WRITETHRU File Handling



For *PREIMG* files opened or created with the *WRITETHRU* attribute, the server behaves as follows:

- *PREIMG* indexes are placed into standard *WRITETHRU* mode except that changes in the number of index entries are output at transaction commit time.
- *PREIMG* data files are placed into a modified mode in which file updates and header updates are only output at transaction commit time.

WRITETHRU minimizes the possibility of lost updates in the event of a catastrophic event because it writes updated cache pages to disk at the time of the update operation. However, *WRITETHRU* does not provide the guarantee of recoverability that *TRNLOG* provides. It is still possible when using *WRITETHRU* for some updates to be lost or for data and index file inconsistencies to exist following a catastrophic event.

Backup and Restore Options for WRITETHRU Files

The backup and restore options for *WRITETHRU* files are the same as for non-*WRITETHRU* files. The distinction is that using *WRITETHRU* can help minimize data loss that could occur due to unwritten updated buffers for *PREIMG* and non-transaction files (although *WRITETHRU* does not provide an absolute guarantee of data/index consistency or recoverable updates as *TRNLOG* does).

When to Use WRITETHRU Files

WRITETHRU is useful for ensuring that updates to data and index files are written to the filesystem cache (or to disk in the case of ISAM updates) at the time of the update (or commit in the case of *PREIMG WRITETHRU* files). *PREIMG* and non-transaction files that do not use *WRITETHRU* can experience significant data loss due to unwritten cached data in the event of an abnormal server termination.

Creating WRITETHRU Files

To create a *WRITETHRU* data or index file, include the *WRITETHRU* filemode in the filemode value specified when creating the file.

- If using an ISAM-level file creation function, include *WRITETHRU* in the *dfilmod* and *ifilmod* fields of the *IFIL* structure supplied to the function.
- If using a low-level file creation function, include *WRITETHRU* in the filemode parameter supplied to the function.

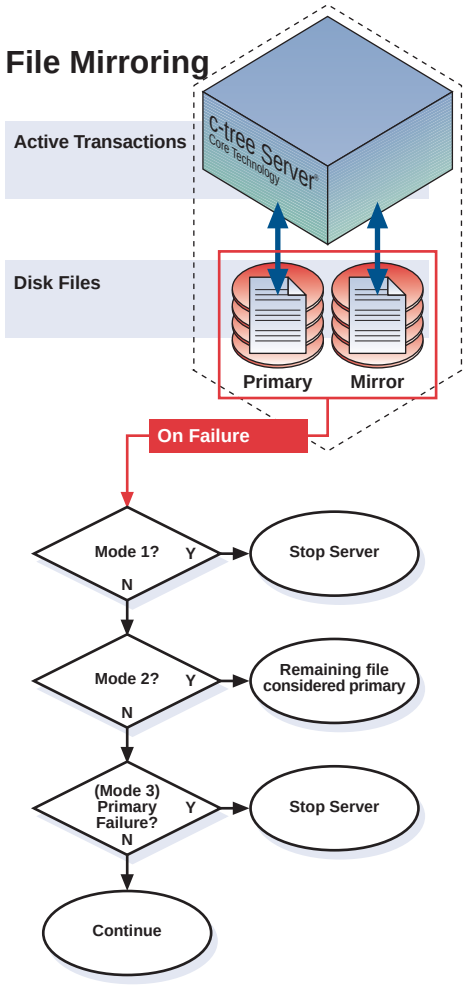
A data or index file that was not created using *WRITETHRU* can be opened as a *WRITETHRU* file by specifying *WRITETHRU* in the filemode when opening the file.

The following server configuration settings can be used to alter the server's WRITETHRU behavior:

- The `COMPATIBILITY FORCE_WRITETHRU` configuration option causes the server to enable *WRITETHRU* semantics for all non-*TRNLOG* data and index files. This option provides a simple way to enable *WRITETHRU* for all files that are not under full transaction control.
- The `COMPATIBILITY WTHRU_UPDFLG` configuration option causes the server to avoid setting the update flag for updated *WRITETHRU* files. This option provides a simple way to avoid error **FCRP_ERR** (14) for *WRITETHRU* files in the event of an abnormal server termination. Note that this option should be used with caution, as an abnormal server termination could leave a data file out of sync with its corresponding indexes (if the data flush completed but the index flush had not yet completed when the server terminated abnormally).

File Mirroring

Figure 11: File Mirroring



The c-tree Server's mirroring facility makes it possible to create and maintain copies of important files on different drive volumes, partitions or physical drives. If the primary storage location is lost due to some form of catastrophe (i.e., head crash) the mirroring logic can automatically detect the lost connection and switch to the secondary or "mirrored" storage area.

By default, read and write operations on mirrored files will continue without returning an error if one of the files fails, but the other succeeds. When this happens, the failed file is shut down, all subsequent I/O operations continue only with the remaining file and the file name for the shut down file is logged in the server status log, *CTSTATUS.FCS*.

Both non-transaction controlled and transaction-controlled files can be mirrored. For a transaction controlled file, if a primary and mirror file get out-of-sync beyond the ability of automatic recovery to make them both good, the most up-to-date file is recovered.

When to Use Mirrored Files

Use mirrored files in a system where disk I/O errors might be expected to occur. Placing a mirrored copy of a file on a different drive than the primary copy of the file can protect against failure of either of the drives. Following a failure of the primary or mirror, the server automatically switches to using only the remaining copy of the file and clients can continue to access the file. Restoring primary and mirror

copies involves taking the remaining copy off-line and using a system file copy utility to restore the second copy of the file from the remaining good copy.

Update speed for mirrored files is no different than for non-mirrored files when the updates are written to the server's cache. When updates are written to disk, however, both the primary and the mirror files are updated, which may cause updates to be slower for a mirrored file compared to a non-mirrored file.

Another option is to use a mirroring approach that operates at the disk level, such as RAID, rather than using the server's mirroring support. Such an approach may provide the following abilities not supported by the c-tree Server's mirroring facility:

- Easily mirroring the entire contents of a disk.
- Maintaining more than two copies of disk contents.
- Better performance due to parallel disk transfer operations.

Creating Mirrored Files

To create a mirrored file, call a c-tree Plus file creation function specifying a name of the form "*<primary>|<mirror>*", where *<primary>* is the name of the primary file and *<mirror>* is the name of the mirror file. The server automatically applies updates made to the primary file to the mirror file.

The c-tree Server also supports mirroring the user/group definition file, *FAIRCOM.FCS*, and the transaction logs and transaction start files. For details on enabling mirroring for these files, see the Mirroring Control options in the "Advanced Configuration Options" section of the *c-tree Server Administrator's Guide.*

Large File Support

When FairCom first released the c-tree Server, 10 MB hard drives were considered large. A 2 GB maximum file size was sufficient, if not unimaginable. For this reason, the c tree Server was originally designed to use 4-byte file offsets, which limited the size of files to 2 GB or 4 GB depending on the platform.

Today, with 64-bit operating systems becoming common and 4 GB hard drives being considered bottom-of-the-line, the need has developed to support very large physical files. Starting with version 7 of the c-tree Server V7, FairCom introduced large file support. The c-tree Server supports large files in two ways:

- Huge files: Physical files that use 8-byte offsets that can grow to over 4 GB in size. Huge files can be used to support large files up to the maximum physical file size supported by modern filesystems.
- Segmented files: Logical files distributed across multiple physical files, which can optionally be huge. Huge segmented files can be used to overcome system limitations on physical file sizes.

Huge Files

A c-tree data or index file that is created with 8-byte offset support is known as a *HUGE* file. A non-*HUGE* file is limited to 2 or 4 GB in size, depending on system file limits. An 8-byte file offset provides a maximum file size of more than 1.844 x 1019 bytes per file, so a *HUGE* file effectively eliminates practical constraints on the file size.

When to Use Huge Files

Use *HUGE* files when the maximum size of a file is expected to exceed 2 GB. The consequence of using a non-*HUGE* file is that the system-imposed 4-byte offset file limit places a hard limit on the supported file size. When a non-huge file reaches the system file size limit, operations that require the

file size to increase (such as adding records or updating variable-length records) will fail. Only deleting records or creating a new file will allow additional records to be added to a non-huge file when it reaches this state. Using huge files avoids this limitation.

Note that even when using huge files, the maximum size of a file the c-tree Server can create is limited by operating system and file system limits on file sizes. For example, on many Unix systems, the ulimit or limit command can set arbitrary limits on the size of a physical file. When developing a system, review operating system and file system file size limits to ensure they support the system's physical file size requirements. If the system's file size limits are too restrictive, consider using the c tree Server's segmented file support (described later) to overcome the file size limitations.

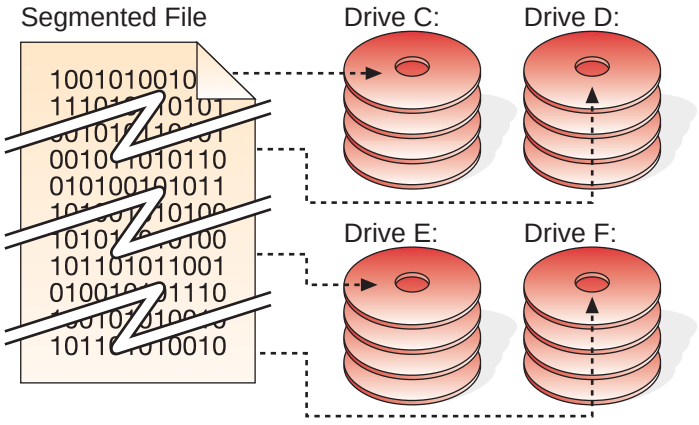
Creating Huge Files

To create *HUGE* data and index files, create an array of *XCREblk* (extended create block) structures, one for each physical data and index file to be created. Include the *ctFILEPOS8* attribute in the *x8mode* field of each extended create block structure. Call an *Xtd8* create function such as **CreateIFileXtd8()**, passing the extended create block array to the function.

Note: Any index referencing a data file created using 8-byte file addresses must also use 8-byte file addresses. A *ctFILEPOS8* data file requires a *ctFILEPOS8* index. A *ctFILEPOS8* index supporting duplicate keys must allow for 8 bytes in its key length for the automatic tie-breaker that c-tree Plus automatically appends to the key value.

Segmented Files

Figure 12: c-tree Segmented Files



Segmented file support allows a single logical file to occupy multiple physical files. This allows data files to exceed the physical file size limits imposed by the operating system, and when combined with Huge File Support, provides the added benefit of very large file sizes. The physical file segments can be kept together or distributed over multiple volumes.

When to Use Segmented Files

Use segmented files in the following situations:

- When the total physical size of a c-tree file is expected to exceed operating system or file system limits. In this case, the file can be segmented into multiple physical files, each of which is within the system's file size limitations.

- When enough total disk space is available for the expected physical file size, but the disk space is spread over multiple volumes. In this case, the file's various segments can be placed where the disk space is available.

Creating Segmented Files

Files can be segmented automatically or the size and location of specific segments can be defined programmatically. Follow these steps to create segmented files:

1. Create an array of extended create block (*XCREblk*) structures, one for each physical file to be created. Include the following flags as appropriate in the *x8mode* field of the *XCREblk* structures:
   a) The *ctFILEPOS8* file mode permits huge files. This mode is required if the logical file will exceed 4GB total file size, but is not required for segmented files in general.
   b) The *ctSEGAUTO* file mode allows automatic segment generation. When this file mode is used, **SetFileSegments()** is not required. Simply set the host segment size and maximum segments (the *segsize* and *segmax* elements of the Extended File Creation Block). All additional segments will be the same size as the host file and will be stored in the same directory. The segment names generated for a file start by adding ".001" to the existing file name, then incrementing the numeric extension. For example: The automatic segment names for the file *sample.dat* would start with *sample.dat.001* and continues with *sample.dat.002*, and so on.
2. Call a c-tree Plus *Xtd8* file creation function, passing it the *XCREblk* structure array.
3. If not using automatic segments, define a *SEGMDEF* structure detailing the segments to be used. To establish the initial segments, call **SetFileSegments()**, passing it the *SEGMDEF* structure.

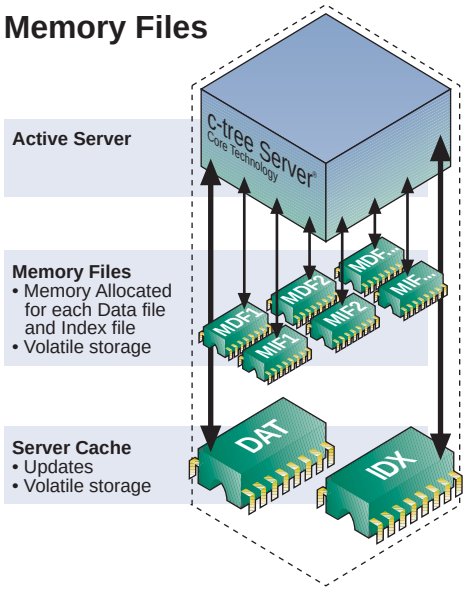
The extended creation functions can only provide minimal information for segmented files. The *XCREblk* structure only holds the size of the host segment and the maximum number of segments. To fill in the details, the **SetFileSegments()** function specifies segment definitions for newly created files dynamically while the file is open. Also, **SetFileSegments()** can optionally set or change the size limit on the host segment. However, **SetFileSegments()** can only be called for files created with the *Xtd8* API, which causes the file to have an extended header.

The *Xtd8* create functions always create Extended files, even if no extended features are requested (unless the *ctNO_XHDRS* file mode is turned on). Files created with the original API (e.g., **CreateIFileXtd()**) are in the Standard c-tree Plus format.

Note: Files are created in *ctEXCLUSIVE* mode, so you must close and reopen the file after it is created to allow it to be shared. Since files are created in *ctEXCLUSIVE* mode, this is a convenient time to execute additional configuration functions, such as **SetFileSegments()** and **PutDODA()**.

Memory Files

Figure 13: c-tree Plus Memory Files



Memory files are data and index files that are purely memory-resident files that do not exist on disk.

The memory in which for memory file data records and index nodes are stored is allocated using the server's internal memory allocation routines and is separate from the server's data and index caches. The server uses its memory management subsystem to allocate and free memory associated with memory files. If the request for memory is at or below 2K bytes, the server uses its own memory sub sub-manager; otherwise, it calls the system memory allocation routine. When a record or node is deleted, the memory is returned.

Properties of Memory Files

Because memory files exist purely in server memory (the data never goes to disk), an abnormal server termination or final close of the file destroys the memory file and its existing memory records.

Memory files can be created as *PREIMG* files or non-transaction files. *PREIMG* memory files support atomic operations just as disk-based *PREIMG* files do.

The maximum size of a memory file is limited by the file size specified when creating the memory file. When selecting memory file sizes, take into account the amount of available physical memory on the system. Creating memory files whose size exceeds the amount of available physical memory on the system will cause the operating system to swap memory to disk, which degrades performance.

A memory resident file cannot be mirrored, partitioned or segmented.

When to Use Memory Files

Memory files are appropriate to use for data that does not need to be stored to disk. Examples include temporary data that is useful during server operation but would normally be discarded when the server is shutdown and restarted and data that can be restored from another source in the event of an abnormal server termination.

Creating Memory Files

To create a c-tree data or index file as a memory file, create an array of *XCREblk* (extended create block) structures, one for each data and host index file to be created. For each extended create block structure, include the *ctMEMFILE* attribute in the *x8mode* field and set the *mxfilzhw* and *mxfilzlw* fields to the maximum file size (*mxfilzhw* is the high-order word and *mxfilzlw* is the low-order word). Call an *Xtd8* create function such as **CreateIFileXtd8()**, passing the extended create block array to the function.

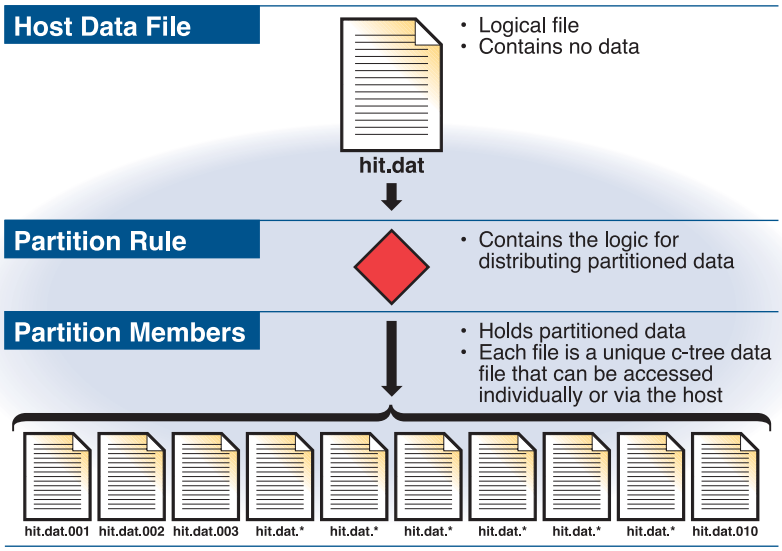
An attempt to add a new record that causes the memory file to exceed this limit fails, and returns a **MAXZ_ERR** (667, attempt to exceed maximum file size).

The final close of a memory file causes the server to delete the file. A final close is a file close that causes the user count of the file to drop to zero. In order to ensure that the contents of a memory file are not lost due to an unintentional close of the file, it is possible to make the data file persist even after the final close by including the *ctKEEPOPEN* bit in the *splval* member of the file's extended create block when creating the file. Then the final close leaves the file open (but without any user attached to the file). It can then be opened again, and the data still exists. The file will be closed when the server terminates, or when a call is made to the c-tree Plus API function **CloseCtFileByName()**.

Partitioned Files

The c-tree Server supports a unique feature known as Partitioned Files. A partitioned file logically appears to be one file (or more accurately one data file and its associated index files), but is actually a set of files whose contents are partitioned by the value of the partition key. Both the data files and index files are partitioned. This permits data with a defined range of values for the partition key to be rapidly purged or archived (instead of having to delete record-by-record each record within this range).

A partitioned file is comprised of a host data file and its associated indices combined with a rule. The rule determines how to map the partition key value (e.g., a date, or invoice number, or some other characteristic) into a particular partition member of the host. The host file does not contain any actual data, but serves as a logical file that appears to contain all the data.



A partition member file has the same definition as the host, but only holds data whose partition key maps to the member. For example, customer orders may be partitioned by calendar quarter. All orders booked in the same quarter are stored in the same member. A member is comprised of a standard c-tree data file and its associated indices.

To add data to the partitioned file, simply call an add routine for the host. The code will add the record in the proper partition member, creating the partition member if necessary. Rewriting a data record may move the record between partitions if the partition key entry for the record is changed. Under transaction control, such a delete/add record operation is done atomically.

Searching the partitioned file in key order is fairly straightforward and efficient if the key is the partition key (the key used to determine which partition should contain the record). Searches on other, non-partition, keys are less efficient than normal because the record may exist in any of the partition members. The more active partition members there are in the logical file, the less efficient the non-partition key search will be.

It is possible to manage the partition members so that archiving or purging partitions is very quick and efficient.

When to Use Partitioned Files

Use partitioned files when the ability to easily purge or archive individual member files is desired. For example, partitioned files can be used to store records by date, so that all records in a given month are stored in one partition. Then, when the month's activity is complete, the partition can be taken offline and archived or purged.

Creating Partitioned Files

To create partitioned files, follow these steps:

1. Activate the partitioned files capabilities at compile time with the *ctPARTITION* define. This define is on by default, provided the *ctHUGEFILE*, *RESOURCE*, *CTS_ISAM*, and *ctCONDIDX* defines are in place. Check *ctree.mak*, *ctoptn.h* and *ctopt2.h* for the current settings.
2. To create a partitioned file using the default partition naming, partition rule, and maximum number of partitions, simply create an Extended file with *ctPARTAUTO* in the *x8mode* parameter of the extended file creation block. Partitioned files do not have to be *HUGE* unless the logical file size will exceed the 2GB/4GB limit, and they also require the extended header, that is, they are Extended files; hence they cannot be accessed by V6 code even when the individual partition file is accessed separately.

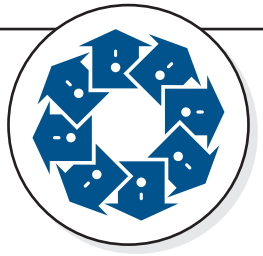
Note that partitioned file support requires a special c-tree Server with application-specific partition rule support. For details, see the c-tree Server SDK documentation and contact FairCom for the required server libraries.

Summary of c-tree Server Features

Below is a summary of the recommended usage of the c-tree Server's features described in this section:

- When designing a system that uses the c-tree Server, first consider using *TRNLOG* for all files, then determine if requirements can be relaxed to include the use of *PREIMG* or non-transaction files in some cases. Understand the effect of using *PREIMG* and non-transaction files.
- Use *TRNLOG* files for critical data whose updates must be fully recoverable in the event of an abnormal server termination. *TRNLOG* provides the best fault tolerance but transaction logging has performance implications.
- If using *TRNLOG* index files, use *LOGIDX* to speed automatic recovery. *LOGIDX* may slightly impact performance but can significantly improve recovery times for large *TRNLOG* indexes.

- Use *PREIMG* files for atomic operations on files without the assurance of recoverable updates. In the event of an abnormal server termination, lost updates are likely for a *PREIMG* file and the file must be re-created, rebuilt, or restored from backup.
- Use non-transaction files for data not requiring atomic operations without the assurance of recoverable updates. In the event of an abnormal server termination, lost updates are likely for a non-transaction file and the file must be re-created, rebuilt, or restored from backup.
- Use *WRITETHRU* (or periodic cache flush calls) to minimize unwritten cached data loss for *PREIMG* and non-transaction files in the event of an abnormal server termination, but be aware of the limitations of *WRITETHRU*: lost updates and out of sync data and index files are still possible, and there is no guarantee of consistency for *PREIMG* files.
- Create c-tree data and index files as huge files when the size of the file is expected to exceed 2 GB. Otherwise, at some point the file will reach its maximum size and updates that would cause the file to grow will fail. Also be aware of system limitations on file sizes (operating system and filesystem settings).
- Create c-tree data and index files as segmented files when the file size is expected to exceed the available space on a single disk volume or when the file size is expected to exceed the supported maximum file size on the system. Otherwise, at some point the file will exhaust available disk space or will exceed the maximum file size and updates that would cause the file to grow will fail.
- Use mirrored files when disk I/O errors are likely to occur. Mirroring protects against the failure of one copy of the file. Another alternative is to use a disk-level mirroring approach such as RAID.
- Use memory files to create memory-based data and index files that exist only while the c tree Server is active. The final close of a memory file frees its memory, destroying its contents. Because memory file data is not persistent, a common use of memory files is to store data that can be reloaded from an external source.
- Use partitioned files when the ability to easily purge or archive portions of a file is desired.



System Implementation and Testing

This chapter describes some recommended scenarios for testing systems that rely upon the c-tree Server. Topics include what aspects of server operation, failure, backup and recovery should be tested, ways to test, and how successful operation can be verified.

2.1 Testing c-tree Server Operation

The primary test focus for a system is upon system operation. This testing involves running the full system with the c-tree Server and observing the behavior of the system in various situations.

Before testing, identify the expected properties of the production system so that the test system can be configured to match the production system as closely as possible. System properties to consider include the following:

- The software and hardware components installed on the system and their configuration (for example, system and application software, and network, disk, memory, and CPU hardware).
- The expected number of c-tree data and index files and the expected sizes of the files.
- The expected real-world data properties (for example, the key value distribution and insertion order affects the organization of the index B+-tree structure).
- The expected system load (number of clients, transactions per second, etc.).

It can be useful to identify not only the expected normal system properties but also their maximum values. Testing system operation in both the normal and the exceptional (maximum load) cases is recommended. A basic testing goal should be to define the normal and maximum loads (based on system specifications and requirements) and to demonstrate through testing that the capacity of the system is sufficient to handle both.

Also consider test cases that involve operations that occur periodically. For example, test system operation during a backup or special end of day processing.

2.2 Testing c-tree Server Backup

System backup is another essential area to test. Consider the following items when testing the system's backup capabilities:

- Test the backup procedure on a system matching the production system configuration as closely as possible.
- Test backup of files matching typical usage and system limits (number of files to backup, size of files to backup)
- Measure the time required to backup the files and determine if it meets system specifications.

- If performing an online backup observe the effect, if any, of the backup on the running system. For example, does the disk and CPU use by the backup process affect the ability of the system to handle its expected load?

2.3 Testing c-tree Server Restore

A system backup is only useful if it can be used to restore the state of a system in the event of a catastrophic failure. After confirming the system's backup abilities, test the system's restore abilities. Consider the following items when testing the system's restore capabilities:

- Measure the time required to restore the files from the backup copy and determine if it meets system specifications. Verify that the restored files are accessible and in the expected state.
- If using forward or backward roll capabilities, measure the time required to perform the forward or backward roll operation and determine if it meets system specifications. Verify the resulting files are in the expected state.
- Test the server's automatic recovery by forcing an abnormal server termination while the system is running a typical load. (The next section describes ways to force an abnormal server termination.) Measure the time required to perform automatic recovery. Verify that the *TRNLOG* files are accessible and are in a consistent transaction state.
- Test rebuilding indexes and compacting data files. Measure the time required to perform these operations and determine if the times meet system specifications. Verify the correctness of the key values and data records after performing rebuild and compact operations.

2.4 Backing Up and Restoring c-tree Files

Data backup and recovery procedures are an important part of a database system. As discussed in Section 2.2, the c-tree Server options used for c-tree data and index files determine the state of the files in the event of a catastrophic failure. These options also determine the available backup options for a file and the resulting state of a file's backup copy. This section discusses how to determine which files to backup, the c tree Server's supported online and offline backup methods, and how these backup and restore features apply to each type of c-tree file.

Determining Files to Include in Backup

The factors that determine which c-tree data and index files should be backed up include:

- The c-tree options in effect for a file: Updates to *TRNLOG* files are made recoverable by transaction logging but this is not the case for *PREIMG* and non-transaction files.
- The data recovery requirements for a file: A system may require updates to a file to be always recoverable. For some files, lost updates might not be a concern or updates could be reapplied from another source.

This section discusses backup considerations for application-created data and index files based on these factors. Backup of server administration files (files the server creates to store user and group definitions and system events) is also discussed.

TRNLOG Files

The c-tree Server guarantees recoverability of *TRNLOG* data and index files by logging updates to *TRNLOG* files to transaction logs on disk. When the server is restarted after a catastrophic failure, the server performs automatic recovery of *TRNLOG* files using the transaction logs. Automatic recovery can be relied upon to recover TRNLOG files to a consistent transaction state provided that the

transaction logs and data and index files are intact after a catastrophic failure. However, if a catastrophic failure damages the transaction logs or the data and index files, automatic recovery might not be able to repair the files. To protect against such a disk failure situation, periodic backups of *TRNLOG* files are recommended.

The c-tree Server's dynamic dump facility can be used to perform a full backup of *TRNLOG* files. The dynamic dump automatically includes the transaction logs and transaction start files in the dump stream file.

Another option is to perform an online disk snapshot (provided the disk subsystem supports this). An online disk snapshot must include the transaction logs, which consist of the following files, in addition to the *TRNLOG* data and index files to be backed up (because automatic recovery must be performed on the files before restoring them):

- *Lnnnnnnn.FCS* (transaction logs)
- *S0000000.FCS* and *S0000001.FCS* (transaction start files)

An offline backup can also be used to backup *TRNLOG* files. An offline backup of *TRNLOG* files does not need to include the transaction logs because the files are closed during the backup.

PREIMG Files

The c-tree Server does not guarantee recoverability of *PREIMG* files in a catastrophic failure situation. For this reason, regular backups of *PREIMG* files are recommended if data must be recovered from the files. If recoverability of a *PREIMG* file is not required (for example, if using a *TRNLOG* data file and a *PREIMG* index and the index can be rebuilt from the recovered contents of the data file, or if a *PREIMG* data file can be re-created and its data re-loaded from an external source), a *PREIMG* file does not need to be backed up.

PREIMG files can be backed up using the server's dynamic dump facility. Specify the name of the *PREIMG* files to backup in the dump script, and use the `PREIMAGE_DUMP` option in the dump script if the files being dumped are in use (being updated) during the dump.

Another way to backup a *PREIMG* file is to perform an offline backup. An offline backup (in which the *PREIMG* files are closed), produces a clean backup of *PREIMG* files.

Non-Transaction Files

The c-tree Server does not guarantee recoverability of non-transaction files in a catastrophic failure situation. For this reason, regular backups of non-transaction files are recommended.

The c-tree Server does not guarantee recoverability of non-transaction files in a catastrophic failure situation. For this reason, regular backups of non-transaction files are recommended if data must be recovered from the files. If recoverability of a non-transaction file is not required (for example, if the potential for losing updates is not a concern, or if a non-transaction data file can be re-created and its data re-loaded from an external source), a non-transaction file does not need to be backed up.

Non-transaction files can be backed up using the server's dynamic dump facility. Specify the name of the non-transaction files to backup in the dump script. If updates occur on a non-transaction file while it is being backed up, the updates are not guaranteed to be included in the backup. Updates can be suppressed for non-transaction files during a dynamic dump by using the `!PROTECT` and `!PROTECT_LOW` keywords in the dynamic dump script. However, even with these options, a non-transaction data file and its corresponding index backed up using a dynamic dump are not guaranteed to be consistent because the files are not dumped at the same time.

Another way to backup a non-transaction file is to perform an offline backup. An offline backup (in which the non-transaction files are closed), produces a clean backup of non-transaction files and ensures the data and index files are consistent.

Server Administration Files

The c-tree Server creates and maintains its own files for server administration purposes. This section describes each of these files and whether or not the file should be included in periodic backups.

Files for Which Backup Is Required

ctsrvr.cfg is the server configuration file. This text file contains the configuration options in effect for the c-tree Server. A backup copy of this file should be saved so that it can be restored if the server configuration file on the production system is lost.

Files for Which Backup May Be Required

CTSTATUS.FCS is the server status log. The file is a text file in which the server logs informational, warning, and error messages. This file does not need to be backed up, although in the event of an abnormal server termination this file may contain error messages that help explain the reason for the abnormal server termination. In such a situation, a copy of the status log should be saved.

FAIRCOM.FCS is a c-tree superfile that holds the c-tree Server's user and group definitions (including user passwords). The data and index members of this file are *TRNLOG* files and are therefore processed by automatic recovery in the event of an abnormal server termination. If the server administrator has defined application-specific users and groups or has changed properties of the ADMIN account, this file should be backed up to guard against catastrophic failures that destroy *FAIRCOM.FCS* or that prevent automatic recovery from completing successfully.

The *Lnnnnnnn.FCS* files (*L0000001.FCS*, *L0000002.FCS*, and so on) are the server's transaction logs. These files contain updates applied to *TRNLOG* files. Automatic recovery uses the transaction logs and the transaction start files *S0000000.FCS* and *S0000001.FCS* to restore *TRNLOG* files to a consistent transaction state after an abnormal server termination. A dynamic dump automatically includes the appropriate transaction logs and start files, and a disk snapshot backup must also include these files to allow automatic recovery of *TRNLOG* files backed up as part of the snapshot. If using the `KEEP_LOGS` option to save inactive transaction logs for use in forward roll operations, the server names inactive transaction logs *Lnnnnnnn.FCA*, and these inactive log files can be safely archived and deleted.

SNAPSHOT.FCS is a text file in which the c-tree Server logs system snapshot statistics. This file is not critical to server operation and does not need to be backed up.

SYSLOGDT.FCS and *SYSLOGIX.FCS* are the server's system log files. They are c-tree data and index files in which the server stores system events. These files are *TRNLOG* files and are therefore processed by automatic recovery in the event of an abnormal server termination. If the system administrator has enabled server event logging and wishes to guard against catastrophic failures that destroy these files or that prevent automatic recovery from completing successfully, these files should be backed up.

Files That Do Not Require Backup

CTSYSCAT.FCS is a c-tree superfile created by the FairCom ODBC Driver for use as a data dictionary. This file contains symbolic names and the names of data files that the ODBC driver exposes to ODBC applications. The contents of this file can generally be re-created from a data dictionary script, so backup of this file is not necessary.

D0000000.FCS is the preimage disk swap file. If the `COMMIT_SWAP` server configuration option is enabled, the server uses this file as necessary for swapping preimage space entries to disk. This file is deleted during a normal server shutdown and can be safely deleted if it exists after an abnormal server termination. This file does not need to be backed up.

D0000001.FCS is a c-tree data file that holds queue entries for the space reclamation of deleted superfile members and recovered variable-length data files (whether in a superfile or stand-alone). The permanent queue permits the space reclamation to be interrupted at server shutdown and resumed when the server restarts. This file is a non-transaction *WRITETHRU* file and does not need to be backed up.

The files *FREOPEN0.FCS*, *FREOPEN1.FCS*, etc. are dummy stream files opened by the c-tree Server at startup. The server uses these files to hold file descriptors for its use when opening stream files. These files are deleted during a normal server shutdown and can be safely deleted if they exist after an abnormal server termination. These files do not need to be backed up.

I0000000.FCS is used during automatic recovery, rollback, and rollfoward operations to store temporary file number to c-tree file number associations. This file is deleted during a normal server shutdown and can be safely deleted if it exists after an abnormal server termination. This file does not need to be backed up.

The c-tree Server stores mappings between file IDs and file numbers in the file *I0000001.FCS*. This file is deleted during a normal server shutdown and can be safely deleted if it exists after an abnormal server termination. This file does not need to be backed up.

I0000002.FCS is the c-tree Server's transaction lock file. At startup, the server creates this file and locks it. If the server is unable to lock this file, the server shuts down. Using this lock file ensures that two servers do not attempt to use the same transaction log directory at the same time. This file is deleted during a normal server shutdown and can be safely deleted if it exists after an abnormal server termination. This file does not need to be backed up.

Online Backup Options

An online backup is a backup taken while the c-tree Server is running and while clients are using the data and index files that are being backed up. An online backup is performed transparently and does not interfere with access to the data. Two options are available for online backup of c-tree data and index files. A dynamic dump is a feature of the c-tree Server that allows transparently backing up a specified set of c tree files to a dump stream file while clients continue to use the files. An online disk backup is a feature supported by some disk subsystems which permits taking a point in time snapshot of disk contents. These options are discussed below.

Dynamic Dump

A dynamic dump provides a safe, secure method of backing up data while the c-tree Server is operational. The server administrator can schedule a dump of specific files, which may be all files necessary for recovery or a subset of them. The dump runs while the Server is carrying on normal activity and processing transactions and is transparent to users.

The Server performs the dump at the first opportunity on or after the scheduled time. When beginning a scheduled dump, the c-tree Server temporarily stops new transactions from beginning and starts the actual dump as soon as all active transactions are complete or after aborting transactions that do not complete during a pre-set delay period. After the dump begins, there are no restrictions on transactions.

The dynamic dump and recovery processes are intended primarily for files under transaction processing control. Non-transaction controlled files can be dumped under certain restrictions, as described in the following sections.

Dumping TRNLOG Files

A dynamic dump automatically includes updates applied to *TRNLOG* files in transaction logs included in the dump stream file. When the files are restored using the **ctrdmp** utility, the restore process

automatically uses the included transaction logs to restore the *TRNLOG* files to a consistent transaction state at the time of the backup.

Dumping PREIMG Files

Although automatic recovery is not available to *PREIMG* files, it is possible to perform periodic dynamic backups. By using `PREIMAGE_DUMP`, it is possible to promote *PREIMG* files to full *TRNLOG* files during the dynamic dump process. The promotion to a *TRNLOG* file means that a full transaction log is maintained during the dump process. This process guarantees that any changes made to the files during the backup are saved in these specially maintained transaction logs. The ability to dynamically backup user data files minimizes the loss of the automatic recovery feature with this mode.

Dumping Non-Transaction Files

It is possible to back up data files that are not under transaction control while the c tree Server remains running. Of course, the safest way to perform a complete backup of data and index files while the c-tree Server remains running is to ensure that all files are under transaction control. This way it is certain that all data and index files are completely synchronized, and updates to the files can continue during a dynamic dump.

Some developers choose not to implement transaction control. In some cases, developers migrating from the c-tree Plus Standalone Multi-user model, *FPUTFGET*, to the c-tree Server, choose to implement the c-tree Server in an *FPUTFGET*-like manner. An *FPUTFGET*-like Server is defined with the following c-tree Server keywords:

```
COMPATIBILITY FORCE_WRITETHRU
COMPATIBILITY WTHRU_UPDFLG
```

Although it is possible to define a non-transaction controlled file within a dynamic dump backup script, there is no protection against updates to this file. In other words, it is possible for the file to be updated during the dynamic dump. Updating a file controlled by transaction processing is okay, because the dump restore process can use the transaction logs to restore to a consistent state. However, updating files NOT under transaction control while they are being backed up presents a situation where there is no known state for the file, resulting in an undefined backup.

The keyword `!PROTECT`, without an argument, when added to a dynamic dump script file causes the non-transaction files to be dumped cleanly by suspending any updates while each file is dumped. The associated index files for a data file are not guaranteed to be consistent with the data file because the files are not dumped at the same time. Updates are only suspended while the data file is being backed up.

This technique ensures the data file is backed up in a known state. The restore process for a non-transaction control file MUST be complemented with an index rebuild. Because protection is for data files only, under most situations, the indexes are not worth dumping since they must be rebuilt.

Note: `!PROTECT` suspends updates at the ISAM level only. The keyword `!PROTECT_LOW` also suspends low-level updates in addition to the ISAM level. FairCom suggests using the `!PROTECT_LOW` when using low-level function calls.

Segmented Dump Stream Files

A dynamic dump stream file can be very large, depending on the sizes of the files being backed up. To avoid operating system file limits, by default the c-tree Server automatically creates a dump stream file as 1 GB-sized extents. For example, dumping 5 GB of data specifying a dump stream file name of backup produces five 1 GB dump stream files with the names *backup*, *backup.001*, *backup.002*, *backup.003*, and *backup.004*. This default can be overridden using the `!EXT_SIZE` dump script option.

The c-tree Server also supports the ability to created segmented dump stream files, in which the dump script defines dump stream file segments and their locations. See the *c-tree Server Administrator's Guide* for details on using the `!SEGMENT` dump script option.

Online Disk Snapshot

Some disk subsystems support taking a disk image snapshot without interfering with the running system. This option, when available, provides a way to backup *TRNLOG* files without losing data or producing an inconsistent transaction state. A disk snapshot does not include the contents of the server's cache, so for non-transaction and *PREIMG* files updates may be lost. Non-transaction and *PREIMG* files restored from a disk snapshot should be re-created or rebuilt. For *TRNLOG* files, the transaction logs contain all committed updates, so the *TRNLOG* files can be restored to a consistent state by restoring the disk image and starting the c-tree Server, allowing it to perform automatic recovery on the *TRNLOG* files using the saved transaction logs.

Offline Backup Options

An offline backup is a backup that is taken in a manner that interrupts the use of some or all of the c-tree Server's data. An offline backup may involve shutting down the c tree Server or allowing the server to run but restricting access to the files as they are backed up. The important point to note is that offline backup options typically involve a file copy operation, so the files to be backed up must be closed before they are copied, in order to ensure that all cached updates have been written to the files. Because the files are closed when copied, this approach allows *TRNLOG*, *PREIMG*, and non-transaction files to be backed up without data loss and maintains transaction consistency for *PREIMG* and *TRNLOG* files.

The general steps in performing an offline backup are as follows:

1. Close the files that are to be backed up.

2. Prevent access to the files being backed up.

3. Back up the files using a system file copy utility.

4. Allow access to the files.

The following sections discuss two offline backup methods and how these four steps are performed in each case.

Offline Backup with Server Shut Down

In some situations, the system is architected in a way that permits the c-tree Server to be shut down in order to perform a backup. For example, a system may be designed to permit another server to take over for a c-tree Server that is shut down, or a system may not be required to be available at all times. In such a situation, c-tree files can be backed up by shutting down the server and copying the data and index files to backup media using system file copy utilities.

This is the simpler of the two offline backup methods. Shutting down the server ensures the files to be backed up are closed (step 1 mentioned above) and prevents access to the files during the backup (step 2). After the backup is complete (step 3), restarting the server allows access to the files again (step 4).

Offline Backup with Server Operational

Another way to perform an offline backup is to take files offline for backup while the c-tree Server remains active. This backup option is appropriate for files that can be closed and backed up without affecting the availability of the system. For example, if using partitioned files, partitions can be taken

offline and backed up. Or, applications can manually maintain multiple sets of files in such a way that some files can be closed and backed up periodically.

This type of offline backup requires the application to be able to control client access to the files to be backed up. While the server continues to run, the files to be backed up must be closed (step 1), access must be prevented to the files as they are being backed up (step 2), and after the backup is complete the files must be made available again (step 4).

To implement these steps, the clients could be designed to close files periodically for backup purposes, or the c tree Server could be used to control file access. The c tree Server provides the ability to control file access through the c-tree Plus **Security()** function. Four modes support the ability of an ADMIN user to block logons by non-ADMIN users and to control internal server threads as shown in the following table:

Mode	Explanation
SEC_BLOCK_NONADM	Block user logons not in ADMIN group
SEC_BLOCK_NONSUP	Block all user logons except ADMIN
SEC_BLOCK_KILL	Block all user logins except ADMIN, and kill all other users (suspend the server)
SEC_BLOCK_OFF	Turn block off

SEC_BLOCK_NONADM and *SEC_BLOCK_NONSUP* can be used to block logon attempts by clients. These options do not affect active client connections but do prevent clients from establishing new connections to the server.

SEC_BLOCK_KILL can be used to effectively suspend the server. This option blocks all user logins except the ADMIN superuser account, and terminates all client connections, causing the server to close all open files and to suspend its internal threads. This option suspends use of the c-tree Server until a *SEC_BLOCK_OFF* call is made.

An application may wish to shut down access to only some files rather than suspending server operation or client logons as described above. In such a situation, the application must implement its own ability to signal clients to close files and to notify them when the files can be reopened. Also, the application must ensure that internal server threads do not access the files as they are being backed up. The ctmove utility can be used to move or rename files in order to ensure that internal server threads do not access a file while it is being backed up. The backup procedure steps for such a situation are as follows:

1. Signal clients to close the files that are to be backed up.

2. Rename or move the files using the ctmove utility.

3. Backup the files using a system file copy utility.

4. Signal clients to reopen the files.

c-tree Server Recovery and Restoration Facilities

In the event of a catastrophic failure such as a disk media failure that renders files unusable, it is necessary to recover data or to restore data from a backup. This section discusses the c tree Server's supported data recovery and restoration options.

When evaluating the available recovery and restoration options, consider the time that will be required to restore the data for the selected approach, such as transaction recovery time, dynamic dump restore time, system rollback or forward roll time.

Recovery of TRNLOG Files Using Automatic Recovery

In the event of an abnormal server termination, *TRNLOG* files can be restored to a consistent transaction state by restarting the c-tree Server. The c-tree Server detects an improper shutdown and performs automatic recovery, applying appropriate changes to *TRNLOG* files based on the activity shown in the server's transaction logs.

If for some reason automatic recovery is not able to complete successfully, *TRNLOG* files may be in an inconsistent state. Periodic backups of *TRNLOG* data and index files can help to guard against such an occurrence.

PREIMG and non-transaction files do not have the benefit of automatic recovery, so in the event of a catastrophic failure, they must be re-created or restored from backup. The following sections discuss data restoration options.

Restoring from Dynamic Dump Backup

Use the **ctrdmp** utility to restore c-tree data and index files from a dynamic dump backup stream file. Observe the following points before running ctrdmp:

1. **ctrdmp** generates temporary versions of transaction logs and other system files associated with a c tree Server (i.e., files with the extension *.FCS*). Therefore, the dynamic dump file and the **ctrdmp** utility should be moved to a directory other than the working directory where the c-tree Server undergoing recovery resides. This is so the system files created by the recovery program will not overwrite working c-tree Server files. The temporary files are automatically deleted when Recovery completes successfully unless `!FORWARD_ROLL` is in the recovery script. In that case, the *S*.FCS* files are renamed to *S*.FCA* and kept in the directory.
2. Be sure the c-tree Server for which files are being recovered is not running when **ctrdmp** starts, or if it is running be sure the files being recovered are not in use by the server.

After taking these preliminary steps, do the following to recover a dynamic dump:

1. Start **ctrdmp** the same way as any normal program in the environment.
2. When prompted, enter the name of the dynamic dump script file to be used for the recovery (or the dump script name can be specified on the command line).

The same script file used to perform the dump can be used to restore the dump. The dump recovery begins automatically and produces a series of messages reporting the progress of the recovery.

1. Each recovered, i.e., recreated, file will be listed as it is completed.
2. After all specified files have been recovered, a message indicates the recovery log (i.e., the transaction log) is being checked and recovered files were restored back to their state as of a given time (i.e., the time the dynamic dump started).
3. **ctrdmp** indicates the dump recovery process finished successfully by writing the message `"DR: Successful Dump Restore Termination" to CTSTATUS.FCS.`

By default, dynamic dump restore creates files in their original directory. The dynamic dump script keyword `!REDIRECT` provides a convenient way to redirect the destination of the data during a dynamic dump restore.

Restoring from Online Disk Snapshot

Systems that support an online disk snapshot provide a way to transparently backup *TRNLOG* files while they are in use. To restore files from an online disk snapshot, the files must first be recovered to a consistent transaction state using the c-tree Server's automatic recovery facility.

To perform automatic recovery on an online disk snapshot, make the files available to the system (by mounting a drive or copying files from the backup media to a system drive). If the application uses full paths when opening data and index files, the files must be placed in the same directory structure as on the production system in order to ensure that automatic recovery can properly open the files.

Start the c-tree Server using the same c-tree Server configuration options used on the production system. The server performs automatic recovery and notes successful completion in its status log, *CTSTATUS.FCS*. Upon successful completion of automatic recovery, the files are ready to use.

Deciding when to perform automatic recovery on the files depends on the system requirements and available resources. One approach is to perform automatic recovery on the files after the backup is taken, so the files are ready for use when needed. This approach has the advantage of minimizing the time taken to restore the system after a catastrophic error, because recovery has been completed in advance. Another approach is to wait until the files must be restored and to run automatic recovery at that time.

Restoring from Offline Backup

An offline backup involves backing up files when they are closed. Files backed up in this manner are in a clean state and can be restored by copying the backup files to the appropriate directory. If overwriting existing files, make sure the files are closed before overwriting the files with the backup copy. After the copy completes, the files are ready for use.

Rolling Forward from Backup

The forward dump utility, **ctfdmp**, can be used to recover from a catastrophic failure following the successful execution of a dynamic dump or from a full backup made after a safe, clean, controlled shutdown of the system.

To prepare for using the forward dump utility, **ctfdmp**, follow these guidelines:

1. Set the `KEEP_LOGS` configuration option to retain all log files. This setting causes log files no longer required for automatic recovery to be renamed instead of deleted. The extensions of log files are changed from *.FCS* to *.FCA*, which changes the transaction log files from “active” to “inactive”. These “old” log files may be needed to roll forward.
2. Make periodic, complete backups using a dynamic dump or offline backup. The following files must be included in a complete backup:
   - All data and index files.
   - The file *FAIRCOM.FCS*.
   - The *S*.FCS* files (automatically included in dynamic dump).
3. Following a safe, complete backup, save all transaction log files created until the next complete backup. Active transaction log files have names of the form *L<log number>.FCS*, with the number being incremented by 1 for each new active transaction log. As specified in the `KEEP_LOGS` configuration value, when the c-tree Server creates a new active log it renames the active log being replaced from *L<log number>.FCS* to *L<log number>.FCA* and saves it as an inactive transaction log file.

If the system has a catastrophic failure and preparations have been made as recommended, the data can be recovered as follows:

1. Restore the contents of the most recent backup, which can be a dynamic dump or a standard backup, provided it includes the files listed in step 2 above.

   Note: If the restore is from a dynamic dump, be sure to include the `!FORWARD_ROLL` keyword in the dump recovery script. This keyword causes creation of a transaction start file for the recovered

logs. The transaction start file will be named *S*.FCA*. After the restore is complete, rename *S*.FCA* to *S*.FCS*.

2. Load all transaction log files saved between the time of that backup and the time of the catastrophic failure and rename all inactive transaction files in this group (i.e., log files with the extension *.FCA*) to give them the extension of an active transaction log file (i.e., extension *.FCS*).
3. Start the forward dump utility, **ctfdmp**, as any other program in the environment. The forward dump will proceed without any further instructions.

Note: Only transaction-processed files will be updated beyond the state they held when the backup was made.

ctfdmp accepts the command line arguments shown below. The first two need to be used only if the application uses more than the default number of `#FCB` or the `PAGE_SIZE` is larger than default. If either of the first two command line arguments is used, they both must be specified as illustrated below. !SKIP is optional and does not cause an error termination if a file required during the forward roll is not accessible. Extreme care must be exercised if `!SKIP` is used, since the forward roll has no way of ensuring the integrity of data for files that are skipped.

```
ctfdmp [!#FCB <files>] [!PAGE_SIZE <page size>] [!SKIP]
```

The **ctfdmp** utility can be used when the c-tree Server is running only if it is run in a directory other than the directory in which the server stores its transaction logs and if the files being rolled forward are not in use by the server.

Rolling Back from Backup

System rollback restores the system to its status as of a specified point in time. For example, if company payroll processing was started at 1:00 PM and something went awry later in the afternoon, a system rollback can reset the system to the way it was at 1:00 PM, so processing could start again. If other applications using transaction processing files were running while the payroll processing was underway, these other files would also be rolled back to their 1:00 PM state. The administrator should be aware of all files and related data that will be affected before starting a rollback to avoid interfering with multiple, unrelated systems sharing a c-tree Server.

Executing a rollback involves running the **ctrdmp** utility using a dynamic dump script with keywords that control how the rollback is to be done. If the first line in the script supplied to **ctrdmp** is `!ROLLBACK`, the script is used for a rollback. Otherwise, the script is considered a dynamic dump script and is used for a dump or a recovery.

Warning: As in dump recovery, be sure that if the particular c-tree Server undergoing the rollback is running, **ctrdmp** is run in a directory other than the server's transaction log directory and that the files it is rolling back are closed.

To prepare for a rollback, follow these guidelines:

1. Set the `KEEP_LOGS` configuration option to retain all log files. This setting causes log files no longer required for automatic recovery to be renamed instead of deleted. The extensions of log files are changed from *.FCS* to *.FCA*, which changes the transaction log files from "active" to "inactive". These "old" log files are needed to roll backward.
2. Make periodic, complete backups using a dynamic dump or offline backup. The following files must be included in a complete backup:
   - All data and index files.
   - The file *FAIRCOM.FCS*.
   - The *S*.FCS* files (automatically included in dynamic dump).

3. Following a safe, complete backup, save all transaction log files created until the next complete backup. Active transaction log files have names of the form *L<log number>.FCS*, with the number being incremented by 1 for each new active transaction log. As specified in the `KEEP_LOGS` configuration value, when the c-tree Server creates a new active log it renames the active log being replaced from *L<log number>.FCS* to *L<log number>.FCA* and saves it as an inactive transaction log file.

If the system has a catastrophic failure and preparations have been made as recommended, the data can be rolled back as follows:

1. Restore the contents of the most recent backup, which can be a dynamic dump or a standard backup, provided it includes the files listed in step 2 above.

   Note: If the restore is from a dynamic dump, be sure to include the `!FORWARD_ROLL` keyword in the dump recovery script. This keyword causes creation of a transaction start file for the recovered logs. The transaction start file will be named *S*.FCA*. After the restore is complete, rename *S*.FCA* to *S*.FCS*.

2. Load all transaction log files saved between the target rollback time and the time of that backup and rename all inactive transaction files in this group (i.e., log files with the extension *.FCA*) to give them the extension of an active transaction log file (i.e., extension *.FCS*).
3. Start the **ctrdmp** utility as any other program in the environment, specifying the name of the rollback script. The rollback will proceed without any further instructions.

Note: Only transaction-processed files will be rolled back to the rollback time.

The rollback produces a series of messages reporting the progress of the rollback. The ctrdmp utility indicates the rollback completed successfully by writing the message:

```
RB: Successful Rollback Termination" to CTSTATUS.FCS
```

and returning an exit code of zero.

Rebuilding Indexes and Compacting Data Files

Depending on the file options in effect for a c-tree index, a catastrophic failure may leave the index in a state that requires the index to be re-created and its key values reconstructed from the data file. This operation is known as rebuilding the index. Because *TRNLOG* indexes are recovered using the server's automatic recovery, they generally do not need to be rebuilt after a catastrophic failure, unless the index file is damaged. *PREIMG* and non-transaction indexes do require rebuilding following a catastrophic failure because any cached updates that had not been written to disk at the time of the failure are lost, and the data and index files may be out of sync.

A file open attempt that fails with error **FCRP_ERR** (14) indicates that the file's update flag is set, meaning that it was updated and not closed properly and could be missing updates or the data and index files could be out of sync. In this situation, the file can be restored to a clean state by rebuilding the indexes.

If a data file and its indexes are created and maintained using c-tree ISAM-level functions, the ISAM file definitions (*IFIL*, *IIDX*, and *ISEG* structures) determine how key values are constructed from data record images. In this case, the **RebuildIFile()**, **RebuildIFileXtd()**, or **RebuildIFileXtd8()** functions or the **ctrbldif** utility can be used to rebuild the indexes using the specified ISAM file definitions. An ISAM rebuild performs the following operations on the data and index files:

1. Resets the update flag in the header of the data file.
2. Rebuilds the internal delete stack chain in fixed-length files and rebuilds the internal delete management index in variable-length files.

3. Removes the existing index file and builds an index, optimized for both size and speed, over the existing data file.
4. If a *SRLSEG* key segment exists, finds the highest serial number in use and updates the file header with that value.

An ISAM rebuild can optionally mark records containing unwanted duplicate key values as deleted. This feature is useful when an index exists that does not allow duplicates and the rebuild unexpectedly finds more than one record in the data file having the same key value. See the *c-tree Plus Function Reference Guide* for details on using this and other ISAM rebuild options.

If a data file and its indexes are created and maintained using c-tree low-level functions, the application determines how key values are constructed from data record images. For this reason, the application must implement its own rebuild functionality. This can be accomplished by writing a utility that reads all records from the data file and for each record constructs key values using the application's key formation logic and adds the key values to the indexes.

The primary purpose of a rebuild is to fix errors that occur in index files. When an error occurs in a data file (for example, due to a catastrophic failure that damages the data file), it may be appropriate to create a new copy of the data file containing the active records that can be read from the damaged data file. This is done by compacting the data file. This procedure is known as compacting because it typically produces a data file smaller than the original data file because the new file contains only active records (no deleted records).

If a data file and its indexes are created and maintained using c-tree ISAM-level functions, the data file can be compacted using the ISAM functions **CompactIFile()** or **CompactIFileXtd()** or the **ctcmpcif** utility. These functions scan the original data file, producing a new data file containing only active records and new optimized indexes. The compact operation automatically calls **RebuildIFile()** to rebuild the indexes after creating the compacted data file. Because the compact produces a new data file, up to twice the data file size is needed to perform this operation.

If a data file and its indexes are created and maintained using c-tree low-level functions, the application must implement its own compact functionality. This can be accomplished by writing a utility that creates a new data file matching the original data file definitions, reads all active records from the data file and for each record constructs key values using the application's key formation logic and adds the key values to the indexes.

c-tree Server Recovery Checklist

Below are recommended steps for server administrators to follow to ensure that the necessary server recovery procedures have been prepared and can be executed in the event of a catastrophic failure:

- Plan procedures for abnormal server termination. Be able to perform automatic recovery on *TRNLOG* files and to return *PREIMG* and non-transaction files to a clean state by re-creating, restoring from backup or rebuilding these files.
- Plan for failed automatic recovery situations. Be able to re-create *TRNLOG* files or to restore *TRNLOG* files from backup if necessary.
- Implement a utility to check the state of *PREIMG* and non-transaction files by attempting to open these files.
- Implement a utility to rebuild files if necessary (**ctrbldif** can be used for this purpose).
- Implement a utility to compact files if necessary (**ctcmpcif** can be used for this purpose).

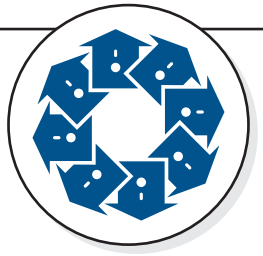
2.5 Testing c-tree Server Failure

Simulating c-tree Server system failure situations is valuable because the results of such tests provide insight into the expected behavior of the system in the event of an actual catastrophic failure. Monitoring the behavior of the system in such cases provides the following benefits:

- Practice using monitoring tools
- Idea of patterns to look for
- Gives idea of state of the system following a catastrophic failure, which influences recovery plans.

Below are some suggested tests that exhaust system resources in an attempt to cause system failure. In each of these cases, use system monitoring tools as described in the next chapter to observe the effect of the test on the system.

- Simulate Saturated CPU: Write a CPU-intensive program and use it to exhaust available CPU resources while the server is running. A likely result of this test is reduced system throughput as threads wait for CPU time. This reduced throughput may trigger second-order effects (such as increased memory use due to increased system queue backlog).
- Simulate Insufficient Disk Space: Write a program that creates large files and use it to exhaust available disk space while the server is running. A likely result of this test is that the c-tree Server shuts down because it is unable to write to its transaction logs.
- Simulate Insufficient Memory: Write a program that allocates large amounts of memory and use it to exhaust available memory while the server is running. A likely result of this test is reduced system throughput due to swapping of virtual memory to disk.
- Simulate Saturated Network: Write a program that performs significant network I/O and use it to consume network bandwidth while the server is running. A likely result of this test is reduced system throughput due to slower communication between clients and the c-tree Server.
- Simulate Abnormal Server Termination: Some of the above tests will cause the server to terminate abnormally. It is also possible to force abnormal server termination directly using the operating system's ability to forcibly terminate a process. For example, the command 'kill -9 <process_id>' can be used on many Unix systems. Use this approach to test automatic recovery and system recovery procedures following an abnormal server termination.



System Operation and Support

Given a running system whose availability depends upon the c-tree Server, the system support team needs to know:

- How to monitor server operation in order to determine the health of the system and to detect when an event occurs (or is about to occur) that threatens the availability or integrity of the database. (Here we can discuss FairCom tools such as **ctstat** and server configuration options and system tools such as **mpstat** and **iostat** to monitor system resource usage.)
- In the case of such an event occurring, how to diagnose the cause of the event, its possible effects, and what to do about it.
- How to perform server operations such as starting and shutting down the c tree Server and backing up and restoring data.

3.1 Starting the c-tree Server

This section describes the required steps for initially setting up the c-tree Server and the steps for starting the c-tree Server.

Initial c-tree Server System Setup

Before discussing the process of starting the c-tree Server, there are a few points to make about the Administrator's first-time duties:

1. Install the c-tree Server as instructed in the *c-tree Server Administrator's Guide*. For Unix systems, server installation typically involves extracting the server binary from a tar file. For Windows systems, server installation typically involves installing using a Windows installer.

   Note: It is possible to run more than one c-tree Server on a system, but doing so requires installing and configuring the servers in a way that ensures they operate in their own working areas without interference. For details, see the "Installing Multiple Instances of the c-tree Server" section of the *c-tree Server Administrator's Guide*.

2. If the c-tree Server is not already activated, activate the server using the c tree Server Activation Utility and the supplied server serial number and activation key.
3. If the application vendor has supplied an (optional) settings file, *ctsrvr.set*, ensure it is in the location specified in the vendor's installation documentation. The settings file is not user-configurable. See the "Advanced Configuration" section in the "Configuring the c-tree Server" chapter of the *c-tree Server Administrator's Guide* for more information.

4. If there is a c-tree Server configuration file, *ctsrvr.cfg*, verify the file contents, change it if necessary, and prepare it to run when the c-tree Server starts. See the "Configuring the c-tree Server" of the *c-tree Server Administrator's Guide* for details.
5. Change the Administrator's password to protect future access to the c-tree Server and access to Administrator utilities. Use the Administrator Utility described in the "c-tree Server Administrator Utility" section of the *c-tree Server Administrator's Guide*.

   Note: Initially, the c-tree Server recognizes only one user, who is intended to be the Administrator. This "super user" has the unchangeable User ID name of ADMIN and the initial password of ADMIN. Administrator functions can be run by anyone with knowledge of the Administrator User ID and password. The first thing to do is to change the initial password and keep the new password secure. The steps to change the password are described in the "c-tree Server Administrator Utility" section of the *c-tree Server Administrator's Guide*.
6. Set up initial User IDs so users can log on to the c-tree Server. Use the Administrator Utility, described in the "c-tree Server Administrator Utility" section of the *c-tree Server Administrator's Guide*.
7. If desired, set access permission on c-tree data and index files and directories used by the c-tree Server to ensure that only the c-tree Server process or c tree utilities executed by authorized users can access the data and index files. Set access permission on the c-tree Server binary to allow execution of the server by only authorized users and to ensure the server can access the necessary configuration, data, and index files.
8. Ensure that system resources are properly configured for the c-tree Server's expected usage. For example, Unix systems typically enforce limits on the number of file descriptors available to a process. Use system utilities such as **ulimit**, limit, or system-specific utilities to configure the system to allow the server process sufficient file descriptors to account for:
   - The maximum number of physical data and index files the c-tree Server is expected to have open at a time (including use of multiple I/O channels; for example using the ctDUPCHANEL file mode), plus
   - A few additional file descriptors to account for the c-tree Server administrative files *CTSTATUS.FCS*, *FAIRCOM.FCS*, *SYSLOGDT.FCS*, *SYSLOGIX.FCS*, and the server's transaction logs (*Lnnnnnn.FCS*, *S0000000.FCS*, and *S0000001.FCS*), plus
   - The maximum number of TCP/IP connections active at a time.

c-tree Server Startup Procedure

The following is a general discussion of the process used to start a c-tree Server. In most environments, **ctsrvr** is the name of the c-tree Server executable. The c-treeSQL Server executable is **ctreesql** and the c-treeSQL Server Java Edition executable is **ctreejql**. In some environments such as Windows, the c-tree Server name may have a file extension (.exe for example). See the platform specific information in the "c-tree Server Installation" chapter of the *c-tree Server Administrator's Guide* for details.

1. Ensure the server binary (and associated DLLs in the case of Windows) are installed in the desired directory. See the "c-tree Server Installation" chapter in the c-*tree Server Administrator's Guide* for platform-specific details.
2. If reconfiguring the c-tree Server, use a text editor to create a configuration file, *ctsrvr.cfg*. See the section "Configuring the c-tree Server" in the c-tree Server Administrator's Guide.
3. Make the server configuration file available to the server by using one of the following approaches:
   a) Place the server configuration file in the c-tree Server's working directory (SYS:\ for NLM), or
   b) Set the FCSRVR_CFG environment variable to the full filename of the server configuration file, or

c) Specify the full filename of the server configuration file on the command line when starting the server (CTSRVR_CFG <file>).

Note: If the configuration file is not found by the c-tree Server, the Server begins operation using default configuration settings. Check the installation instructions for your platform in the "c-tree Server Installation" chapter of the *c-tree Server Administrator's Guide* for any exceptions.

4. Start the c-tree Server by entering or selecting the name of the c-tree Server executable file, **ctsrvr**, just as any ordinary program in the environment.

Note: No password is required to start the c-tree Server; therefore an automated process, such as a batch, script, or cron process, may start the c tree Server.

Every time the c-tree Server starts, it checks log files made when it last stopped and, if necessary, uses these files to automatically recover from problems. See "Automatic Recovery" in the *c-tree Server Administrator's Guide* for details.

Starting the c-tree Server on Unix Systems

In Unix environments, FairCom recommends administrators run the c-tree Server in background to prevent unwittingly terminating the c-tree Server and using the **nohup** utility to ensure the c-tree Server keeps running when the user that started the c-tree Server logs off the system. For example, the following command starts the c-tree Server in background using **nohup** and redirects the server's console output to the file *ctsrvr.log*:

```
nohup ctsrvr > ctsrvr.log &
```

Starting the c-tree Server on Windows Systems

In Windows environments, the c-tree Server may be run as a standard process or as a Windows service. To run the c-tree Server as a standard process, launch the server binary (**ctsrvr.exe**) using Windows Explorer or from the command line.

To install the c-tree Server as a Windows service, use the **ctntinst** utility as described in the *c-tree Server Administrator's Guide*. After the server is installed as a Windows service, it can be started using the **ctntinst** utility, the Windows Service Control Manager, or the 'net start' utility.

3.2 Safely Copying c-tree Server Controlled Files

Warning: c-tree Server controlled files should only be copied, moved, or deleted when the c-tree Server is shut down. Copying, moving, or deleting files while the c tree Server is in operation can lead to unpredictable errors.

The following are important points for c-tree Server administrators to observe:

- Do not use system copy utilities to copy c-tree data and index files or server administration files (**.FCS*) while they are in use by the c-tree Server. To safely copy files while the server is operational follow the approaches discussed in the "Online Backup Options" in Section 2.3 of this document.
- Do not use third-party file scan or backup utilities on c-tree data and index files or server administration files (**.FCS*) while they are in use by the c-tree Server.

Failing to observe the above recommendations could prevent the c-tree Server from accessing the files, which could lead to errors returned to client applications or could cause the server to terminate abnormally. Consider setting file permissions on the c tree data and index files and server administrative files to ensure that only the c-tree Server can access these files while the server is running.

Administrators should also be aware of the use of file IDs in the header of c-tree data and index files. When a file open is attempted, the c-tree Server checks to see if either a file with the same name has already been opened, or if a file with the same unique ID has already been opened. In either case, the match means that a physical file open is not required. Instead the open count for the file is incremented. Checking the unique file ID permits different applications and/or client nodes to refer to the same file with different names: maybe different clients have different drive or path mappings.

However, if two different files have the same ID (a 12-byte value comprised of a Server ID, a time stamp, and a transaction log sequence number), problems could arise because the second file would not actually be opened. The ID is constructed so that no two files could have the same ID unless someone copies one file on top of another. See the warning listed below.

When a file without a matching name does match the unique file ID, a message is sent to the system console indicating the names of the two files involved. This message can be suppressed by adding the following entry to the c-tree Server configuration file:

```
MONITOR_MASK    MATCH_FILE_ID
```

Warning: As discussed above, copying a file to a new name is typically the only way the file ID's can match. If this becomes necessary (i.e., Only to be done when the server is stopped. Do NOT copy, move, or delete files controlled by the server while the server is operational!), choose from the following utility or function:

- UTILITY: *newid.c* - is an informal utility that allows you to change the c tree Server file ID value in a specific file's header. When a file is created by the c-tree Server, a Server ID is placed in the header of the file. This is used by the c-tree Server as a unique reference. If you want to copy a file to a different name, and use them both under the same c-tree Server, you need to change this file ID in one of the files.
- FUNCTION CALL: *updateid.c* - Same as the utility above in function call form.

3.3 Shutting Down the c-tree Server

The c-tree Server can be stopped in any of the following ways:

- Using the **ctadmn** utility
- Using the **ctstop** utility
- Using the Shutdown menu item on the Control menu (Windows server only).
- Using the Windows Service Control Manager or 'net stop' command (if server is run as a Windows service)
- Using the **StopServer()** or **StopServerXtd()** function from a client application
- Using system utilities to terminate the server process

The following sections discuss these shutdown methods.

Shutting Down the c-tree Server Using the ctadmn Utility

The **ctadmn** utility is a client application that can be used to perform various server administration tasks, including shutting down the c-tree Server. To shut down the server using **ctadmn**, follow these steps:

1. Start **ctadmn**.
2. Enter the following information when prompted:
   a) The user ID of a member of the ADMIN group.
   b) The ADMIN password.

c) The optional *FAIRCOM.FCS* file password.

d) The name of the c-tree Server. Specify the server name in the proper format for the communication protocol **ctadmn** is using (for example, server_name@ip_address if using TCP/IP).

3. **ctadmn** connects to the c-tree Server and displays its main menu. Enter '7' to select the "Stop Server" option.
4. Enter YES when prompted to confirm that you wish to shut down the server.
5. Enter the server shutdown delay (press RETURN for no delay).
6. If clients are logged in, **ctadmn** shows the number of connected clients and prompts to confirm that you wish to shut down the server. Enter YES to proceed. **ctadmn** initiates server shutdown and disconnects from the server.

Shutting Down the c-tree Server Using the ctstop Utility

The **ctstop** utility is a client application that can be used to shut down the c-tree Server. To shut down the server using **ctstop**, follow these steps:

1. Start **ctstop**.
2. Enter the following information when prompted:

a) The user ID of a member of the ADMIN group.

b) The ADMIN password.

c) The name of the c-tree Server. Specify the server name in the proper format for the communication protocol **ctstop** is using (for example, server_name@ip_address if using TCP/IP).

d) The delay time (if any) before shutting down the c-tree Server. If a greater than zero delay is specified, the c-tree Server will not accept any new users or transactions. Logon attempts during the delay time specified will fail with c-tree error **SHUT_ERR** (150, Server is shutting down). New transactions cannot be started while waiting to shut down. An attempt to begin a transaction will fail with c-tree error 150 or error **SGON_ERR** (162, Server has gone away), depending on how far the shutdown process has gone.

Shutting Down the c-tree Server for Windows Using the Control Menu

The c-tree Server for Windows can be shut down by selecting the Shutdown menu item on the server's Control menu, or by clicking the Close button at the upper right corner of the main server window. Depending on the server configuration options specified, the server may prompt for the user ID and password of a member of the ADMIN group. If there are connected clients, the server prompts the administrator to confirm the shutdown operation.

Figure 14: Shutting Down the c-tree Server Using the Control Menu

Figure 15: Shutting Down the c-tree Server by Clicking the Close Button

Note: When the `CONSOLE TOOL_TRAY` option is specified in the server configuration file, the server's windows are hidden at startup and a server icon is added to the system tray. Right-clicking the server icon in the system tray produces a menu with the options "Open the FairCom Server's Console" (which can be used to make the server's windows visible) and "ShutDown the FairCom Server" which can be used to shutdown the c-tree Server. When the server is run in this mode, clicking the Close

button at the upper right corner of the main server window hides the server windows rather than causing the server to shut down.

Figure 16: Shutting Down the c-tree Server Using the System Tray Menu

Shutting Down the c-tree Server When Run as a Windows Service

When the c-tree Server is run as a Windows service, the server service can be stopped using the Windows Service Control Manager, the **ctntinst** utility, or the 'net stop' command. Any system user who has permission to stop the server service can stop the c-tree Server in this fashion. The server does not prompt for an ADMIN password in this type of shutdown. For details on these methods of shutting down the c-tree Server service, see the "Stopping the c-tree Server Service" section of the c-tree Server Administrator's Guide.

Shutting Down the c-tree Server Using the StopServer Function

The c-tree Server may be stopped by a client by calling the **StopServer()** or **StopServerXtd()** c-tree Plus API functions. These functions require the password of a member of the ADMIN group. See the *c-tree Plus Programmer's Reference Guide* for details on using these functions.

Shutting Down the c-tree Server Using System Utilities

The above procedures are the recommended procedures for shutting down the c-tree Server process. However, it is also possible in some cases to use system utilities to shut down the c-tree Server. On Unix systems, the 'kill' utility can be used by authorized users to send a SIGTERM signal to the server, which causes the server to initiate a normal shutdown. For example:

```
kill <server_process_id>
```

Emergency c-tree Server Shutdown Using System Utilities

If the c-tree Server must be shut down but it does not respond to the normal shutdown procedures described above, the c-tree Server process can be forcibly terminated using system utilities. This type of server shutdown is for use only in emergency situations because forcibly terminating the c-tree Server process can lead to data loss for non-*TRNLOG* files and typically requires automatic recovery to occur for *TRNLOG* files during the next c-tree Server startup.

A hard kill of the server process can be performed on Unix systems using the command:

```
kill -9 <server_process_id>
```

and can be performed on Windows systems using the Task Manager's "End Process" option.

Additional c-tree Server Shutdown Details

During c-tree Server shutdown, messages reflect when communications terminate and when the final system checkpoint begins. In addition, two aspects of the shutdown that involve loops with two-second delays generate output indicating their status. The first loop permits the delete node queue to be

worked down. The second loop permits clients to shutdown cleanly during c-tree Server shutdown (which can involve flushing of updated cache pages to disk). If these loops are entered, the c-tree Server could take a measurable amount of time to shut down, depending on the amount of work to be done, and output indicates how many queue entries or clients remain. A notice indicates whether everything was cleaned-up. A clean-up notice is NOT generated if a loop was not entered.

This output permits a c-tree Server Administrator to monitor the shutdown, and avoid an incorrect assumption about whether the c-tree Server is making progress or has hung during shutdown. After the c-tree Server shuts down, it sends a message saying c-tree Server operations have been terminated. The output is routed to the console and *CTSTATUS.FCS*, although the latter does not receive the numeric information concerning the number of queue entries or active clients.

3.4 Performing c-tree Server Backup and Recovery Operations

The c-tree Server administrator should understand how to backup and recover system data. This section describes available backup and restore operations for c-tree Server data.

Backup Procedures

This section discusses available c-tree Server data backup procedures.

Dynamic Dump Backup

To schedule a dynamic dump, create a dump script containing the desired options and listing the files to be backed up and use the **ctdump** utility or specify the DUMP keyword in the server configuration file.

For full details about scheduling dynamic dump backups, see the "Dynamic Dump" section in the "Maintaining Database Integrity" chapter of the *c-tree Server Administrator's Guide*.

Online Disk Snapshot

To perform an online disk snapshot of c-tree *TRNLOG* data and index files, use a system utility to take an instantaneous snapshot of c-tree data and index files and the server's transaction logs. Before using the backed up files, start the c-tree Server with the data and index files and transaction logs in place so the server can perform automatic recovery on the files.

Offline Backup

To perform an offline backup of c-tree data and index files, proceed as described in the "Offline Backup Options" section of this document.

Recovery and Restore Procedures

This section discusses available c-tree Server data recovery and restore procedures.

Recovering Using Automatic Recovery

The c-tree Server uses automatic recovery to ensure that *TRNLOG* files are in a consistent transaction state following an abnormal server termination. To use automatic recovery, simply start the c-tree Server as usual. During server startup, the server checks the transaction logs and determines whether or not automatic recovery is needed. If so, the server performs recovery and upon successful completion the server continues its normal startup procedure.

The keyword `RECOVER_DETAILS YES` can be specified in the c-tree Server configuration file to cause the server to log details about the progress of automatic recovery to the server status log, *CTSTATUS.FCS*.

Recovering Using Rebuild and Compact

File rebuild and compact operations can be used to ensure data and index files are in sync and to correct errors that occur when opening, reading, or writing c-tree data and index files. For examples of such error situations, see the sections "Errors Occur When Opening c-tree Files" and "Errors Occur When Reading or Writing c tree files" in the troubleshooting chapter of this document.

For details on options for rebuilding indexes and compacting data files, see the "Rebuilding Indexes and Compacting Data Files" section of this document.

Restoring from Dynamic Dump Backup

Use the **ctrdmp** utility to restore from a dynamic dump backup. For details on using this utility, see the section "Restoring from Dynamic Dump Backup" of this document.

Restoring Using Forward Roll or Rollback

The **ctfdmp** utility can be used to roll *TRNLOG* files forward from a backup to a later point in time provided that the transaction logs are saved from the time of the backup to the target time. The **ctrdmp** utility can be used to roll *TRNLOG* files backward from a backup to an earlier point in time provided that the transaction logs are saved from the target time to the time of the backup.

For details on rolling *TRNLOG* files forward or backward, see the sections "Rolling Forward from Backup" and "Rolling Back from Backup" in Chapter 2 of this document.

Restoring from Online Disk Snapshot

Restoring from an online disk snapshot is accomplished as following:

1. Restore the c-tree *TRNLOG* data and index files and the server's transaction logs from the disk snapshot.
2. Open all *PREIMG* and non-transaction files included in the online disk snapshot and re-create or rebuild those for which the open fails with c-tree error 14.
3. Start the server so that it can perform automatic recovery on the *TRNLOG* files.

Restoring from Offline Backup

Restoring from an offline backup is accomplished simply by copying the backed up files to their appropriate locations on the system. The files cannot be in use while they are being restored.

Verifying the State of the System After Restoration

After restoring files, verify the state of the system and its data. Some ways to accomplish this are:

1. Confirm that all c-tree data and index files can be opened.
2. Test common application functions on the files (reading/writing data).
3. Verify that the data is in the expected state (files contain the expected number of records and data looks correct, the expected relationships between data in multiple files are maintained, etc.).

4. Monitor the c-tree Server's operation as usual (including the c-tree Server status log and application logs) to detect whether there is any abnormal behavior after restoring data.

3.5 Monitoring the c-tree Server and Its Data

Monitoring a system provides system administrators feedback on the health of the system. Knowing the expected resource usage of a system during normal operation and tracking the resource usage of a system over time can give advance warning of events that threaten system availability or data integrity. This section identifies system resources and internal c tree Server resources to monitor and the appropriate tools to use to monitor these resources.

Monitoring System Resource Usage

The c-tree Server relies upon system resources such as the CPU, disk, memory and the network. Typically each system includes its own tools for monitoring system resources. The following sections provide an overview of system-specific monitoring tools for system resources.

Monitoring CPU Usage

The system administrator should know the expected pattern of CPU use by the c-tree Server during normal operation of the system. CPU metrics such as user time, system time, I/O wait time, idle time, and context switches for the server process should be tracked so that unexpected changes in CPU usage can be detected, analyzed, and corrected.

Solaris supports the following utilities for monitoring CPU usage. Other Unix systems support these and similar utilities:

- *mpstat* reports per-processor statistics including counts for interrupts, context switches, spins on mutexes and reader/writer locks, system calls, and user/system/wait/idle times.
- *prstat* reports active process statistics including process state, priority, number of lightweight processes, and CPU usage.
- *top* displays the top processes on the system and periodically updates this information. Raw CPU percentage is used to rank the processes.
- *ps* prints information about active processes.

The Windows Performance Monitor utility (*perfmon*) can be used to monitor CPU usage. The *Processor* performance object maintains counters for each CPU. The available counters include the following (descriptions taken from the Performance Monitor's explanatory text):

- *% Processor Time* is the percentage of elapsed time that the processor spends to execute a non-Idle thread. It is calculated by measuring the duration of the idle thread is active in the sample interval, and subtracting that time from interval duration. (Each processor has an idle thread that consumes cycles when no other threads are ready to run). This counter is the primary indicator of processor activity, and displays the average percentage of busy time observed during the sample interval. It is calculated by monitoring the time that the service is inactive, and subtracting that value from 100%.
- *% Privileged Time* is the percentage of elapsed time that the process threads spent executing code in privileged mode. When a Windows system service in called, the service will often run in privileged mode to gain access to system-private data. Such data is protected from access by threads executing in user mode. Calls to the system can be explicit or implicit, such as page faults or interrupts. Unlike some early operating systems, Windows uses process boundaries for subsystem protection in addition to the traditional protection of user and privileged modes. Some work done by Windows on behalf of the application might appear in other subsystem processes in addition to the privileged time in the process.

- *% User Time* is the percentage of elapsed time the processor spends in the user mode. User mode is a restricted processing mode designed for applications, environment subsystems, and integral subsystems. The alternative, privileged mode, is designed for operating system components and allows direct access to hardware and all memory. The operating system switches application threads to privileged mode to access operating system services. This counter displays the average busy time as a percentage of the sample time.

Monitoring Disk Usage

The system administrator should know the expected pattern of disk use by the c-tree Server during normal operation of the system. Expected data and index file sizes and disk I/O should be tracked so that unexpected changes in disk usage can be detected, analyzed, and corrected.

Solaris supports the following utilities for monitoring disk usage. Other Unix systems support these and similar utilities:

- *vmstat* reports virtual memory statistics regarding process, virtual memory, disk, trap, and CPU activity.
- *iostat* iteratively reports terminal, disk, and tape I/O activity, as well as CPU utilization.

The Windows Performance Monitor utility (*perfmon*) can be used to monitor system disk usage. The *PhysicalDisk* performance object maintains counters for each physical disk. The available counters include the following (descriptions taken from the Performance Monitor's explanatory text):

- *% Disk Read Time* is the percentage of elapsed time that the selected disk drive was busy servicing read requests.
- *% Disk Write Time* is the percentage of elapsed time that the selected disk drive was busy servicing write requests.
- *% Idle Time* reports the percentage of time during the sample interval that the disk was idle.
- *Avg. Disk Queue Length* is the average number of both read and write requests that were queued for the selected disk during the sample interval.

The *LogicalDisk* performance object maintains counters for each logical disk. The available counters include the following:

- *Free Megabytes* displays the unallocated space, in megabytes, on the disk drive in megabytes. One megabyte is equal to 1,048,576 bytes.
- *Current Disk Queue Length* is the number of requests outstanding on the disk at the time the performance data is collected. It also includes requests in service at the time of the collection. This is a instantaneous snapshot, not an average over the time interval. Multi-spindle disk devices can have multiple requests that are active at one time, but other concurrent requests are awaiting service. This counter might reflect a transitory high or low queue length, but if there is a sustained load on the disk drive, it is likely that this will be consistently high. Requests experience delays proportional to the length of this queue minus the number of spindles on the disks. For good performance, this difference should average less than two.

The *Cache* performance object maintains counters for the file system cache. The available counters include the following:

- *Data Flushes/sec* is the rate at which the file system cache has flushed its contents to disk as the result of a request to flush or to satisfy a write-through file write request. More than one page can be transferred on each flush operation.
- *Lazy Write Flushes/sec* is the rate at which the Lazy Writer thread has written to disk. Lazy Writing is the process of updating the disk after the page has been changed in memory, so that the application that changed the file does not have to wait for the disk write to be complete before proceeding. More than one page can be transferred by each write operation.

In addition to system tools available for monitoring disk usage, the c-tree Server supports options to limit disk usage. The server's Disk Full feature offers three levels of control over disk full checks:

- The `DISK_FULL_LIMIT` keyword provides checking on all files.
- The `DISK_FULL_VOLUME` keyword sets a volume-specific limit that overrides the system-wide limit.
- The file-specific check (set by creating a file using an Xtd8 create function with the *dskful* member of the *XCREblk* structure set to the desired file size limit in bytes) overrides the system-wide and volume-specific checks.

If extending the size of the file would leave less than the specified threshold, then the write operation causing the file extension fails, returning **SAVL_ERR** (583).

Monitoring Memory Usage

The system administrator should know the expected memory use by the c-tree Server during normal operation of the system. Memory use should be tracked so that unexpected changes in memory usage can be detected, analyzed, and corrected.

Solaris supports the following utility for monitoring memory usage. Other Unix systems support similar utilities:

- *vmstat* reports virtual memory statistics regarding process, virtual memory, disk, trap, and CPU activity.

The Windows Performance Monitor utility (*perfmon*) can be used to monitor system memory usage. The Memory performance object maintains counters for memory usage. The available counters include the following (descriptions taken from the Performance Monitor's explanatory text):

- *AvailableMBytes* is the amount of physical memory available to processes running on the computer, in Megabytes, rather than bytes as reported in Memory\\Available Bytes. It is calculated by adding the amount of space on the Zeroed, Free, and Stand by memory lists. Free memory is ready for use; Zeroed memory are pages of memory filled with zeros to prevent later processes from seeing data used by a previous process; Standby memory is memory removed from a process' working set (its physical memory) on route to disk, but is still available to be recalled. This counter displays the last observed value only; it is not an average.
- *Pages/sec* is the rate at which pages are read from or written to disk to resolve hard page faults. This counter is a primary indicator of the kinds of faults that cause system-wide delays. It is the sum of Memory\\Pages Input/sec and Memory\\Pages Output/sec. It is counted in numbers of pages, so it can be compared to other counts of pages, such as Memory\\Page Faults/sec, without conversion. It includes pages retrieved to satisfy faults in the file system cache (usually requested by applications) non-cached mapped memory files.

In addition to the available system monitoring tools, the c-tree Server supports monitoring server memory usage using the c-tree Plus **SystemConfiguration()** API function. See the section "Monitoring c-tree Server Using SystemConfiguration API" for more details.

Monitoring Network Usage

The system administrator should know the expected network use by the c-tree Server during normal operation of the system. Network use should be tracked so that unexpected changes in network usage can be detected, analyzed, and corrected.

Solaris supports the following utility for monitoring network usage. Other Unix systems support similar utilities:

- *netstat* displays the contents of network-related data structures in various formats, depending on the specified options.

The Windows Performance Monitor utility (*perfmon*) can be used to network usage. The Network Interface performance object maintains counters for network usage. The available counters include the following (descriptions taken from the Performance Monitor's explanatory text):

- *Bytes Received/sec* is the rate at which bytes are received over each network adapter, including framing characters. Network Interface\\Bytes Received/sec is a subset of Network Interface\\Bytes Total/sec.
- *Bytes Sent/sec* is the rate at which bytes are sent over each each network adapter, including framing characters. Network Interface\\Bytes Sent/sec is a subset of Network Interface\\Bytes Total/sec.
- *Output Queue Length* is the length of the output packet queue (in packets). If this is longer than two, there are delays and the bottleneck should be found and eliminated, if possible. Since the requests are queued by the Network Driver Interface Specification (NDIS) in this implementation, this will always be 0.
- *Packets Outbound Errors* is the number of outbound packets that could not be transmitted because of errors.
- *Packets Received Errors* is the number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol.
- *Packets Received/sec* is the rate at which packets are received on the network interface.
- *Packets Sent/sec* is the rate at which packets are sent on the network interface.

Other System Monitoring Options

In addition to the system monitoring utilities described in the above sections that are target specific resource monitoring, there are also other system utilities that can be used to monitor a variety of system resources.

On Solaris and Unix systems, the *sar* utility is useful for monitoring system activity, including system buffer transfer activity, cache hit ratios, system calls, and device activity.

The Windows Performance Monitor includes performance objects such as the Object, Process, Server, System, and Thread objects that can be used to monitor system resource usage in specific ways. Also, the Windows Task Manager can be used to monitor system resource usage.

Monitoring c-tree Server Internal Resource Usage

The c-tree Server supports utilities and APIs that can be used to monitor its internal resources (such as data and index caches, lock table, transaction and file activity). The following sections explain how to use these utilities and APIs to monitor the server's internal resources.

Monitoring c-tree Server Using Snapshot Support

The c-tree Server's Performance Snapshot capability enables the capture of performance monitoring data using a combination of configuration file options and calls to the **SnapShot()** function. The performance data can be captured automatically at specified intervals or on demand.

Performance data maintained and made available by the server's snapshot capability includes the following: File lock statistics, transaction time histogram, transaction commit delay statistics, transaction log I/O, function timings, data and index cache details, disk read and write statistics, and communication read and write statistics.

The c-tree Server supports the snapshot capability on systems that support 8-byte integers. On systems that do not support a high-resolution timer, some aspects of the snapshot are not available.

Snapshot Configuration File Options

Server configuration file options can be used to capture automatic system snapshot data.

When `SNAPSHOT_INTERVAL <interval in minutes>` is specified in the server configuration file, an automatic snapshot thread performs a system snapshot after each interval. The automatic system snapshot is written to the c-tree Server's system event log (*SYSLOG*) files.

The following configuration entries can be used multiple times to include snapshot details to for users and files in the snapshot output. Use a pattern of '*' to activate monitoring for all user ID's or files.

```
SNAPSHOT_USERID <user ID>
SNAPSHOT_FILENAME <file name>
```

Specifying `DIAGNOSTICS SNAPSHOT_SHUTDOWN` in the server configuration file causes the server to log system snapshot details to the file *SNAPSHOT.FCS* when the server shuts down.

Snapshot API Function Options

The **SnapShot()** API function can be used to control automatic snapshots, overriding configuration options (if any), and can capture on-demand server performance data to the *SYSLOG* files, to the human-readable *SNAPSHOT.FCS* file, or as a return to the calling program. See the c-tree Plus Function Reference Guide for full details on using the **SnapShot()** API function.

Monitoring c-tree Server Using ctstat Utility

The **ctstat** utility is a client utility used to display statistics collected by the c-tree Server. Depending on the specified command-line options, **ctstat** displays file, system, user, or function timing statistics. **ctstat** uses the *SnapShot* API function to log server statistics to the server's system event log, then reads the statistics from the logs and displays them in human-readable format.

See "ctstat Utility Reference" in this document for full details on using the **ctstat** utility.

Monitoring c-tree Server Using SystemConfiguration API

The c-tree Plus API function **SystemConfiguration()** returns an array of LONGs describing the c-tree Server configuration, as well as some of the important dynamic aspects of the system such as the memory usage an the number of files in use. The following sections list **SystemConfiguration()** return values relevant to specific types of monitoring such as server lock table, file usage, and cache usage. For a detailed list of the values returned by **SystemConfiguration()**, see the *c-tree Plus Function Reference Guide*.

Monitoring c-tree Server Memory Use

The **SystemConfiguration** function can be used to display current c-tree Server memory usage figures as shown in the following table:

Array Subscript	Explanation
cfgMEMORY_USAGE	Current system memory usage.
cfgMEMORY_HIGH	Highest system memory use.
cfgNET_ALLOCS	Current system net allocations.

Monitoring c-tree Server Lock Table

The c-tree Server uses a memory-resident lock table for managing locks on c-tree data records. The following sections discuss options for monitoring the state of the server's lock table.

LockDump API Options

The c-tree Plus API function **LockDump()** creates a diagnostic dump of the c-tree Server internal lock table. Locks can be dumped for a particular file or user or for all files in the system.

For a c-tree Server with snapshot support enabled, the **LockDump()** function also logs disk I/O statistics for each file included in the lock dump. These disk I/O statistics consist of the number of read and write operations and bytes read and written for each file.

Below is sample lock dump output for the c-tree data file *mark.dat*:

```
----------------
----------------
mark.dat>>
        0000-00094480x T013 write/1: W18 W19
        0000-00094580x T014 write/1
        0000-00094200x T010 write/1
        0000-00094600x T012 write/1

cumulative lock attempts: 9462(4731)  blocked: 0(0)  dead-lock: 0  denied: 0

Current file lock count: 4
----------------

cumulative I/O: read ops: 76 bytes: 614528    write ops: 38 bytes: 614400
```

The output shows four write locks held on *mark.dat*. The log lists the following details for each lock currently held on a file:

- The record offset of the locked record (0000-00094480x)
- The ID of the thread that is holding the lock (T013)
- The type of lock (write/1)
- The IDs of any threads that are waiting to acquire the lock (W018 W019)

The possible lock types are shown in the following table. Note that the first 5 lock types in the table are supported only for a c-tree Server built with strict serialization support:

Lock Type	Explanation
SS open	SS (strict serializer) logical Open lock
SS commit intent	SS commit intent lock
SS commit	SS commit lock
NS commit intent	NS (nonstrict serializer) commit intent lock
NS commit	NS commit lock
read	Read lock
write/1	Exclusive write lock
write/2	Exclusive write lock (no aggregate check)

SnapShot API Options

The **SnapShot()** function can be used to retrieve lock details. See the discussion of the *SNAPSHOT.FCS* file format for details.

SystemConfiguration API Options

The **SystemConfiguration()** function can be used to display the current number of pending locks:

Array Subscript	Explanation
cfgNET_LOCKS	Current number of pending locks (system wide).

Monitoring c-tree Client Activity

The c-tree Server provides a variety of ways to monitor c-tree client activity. The following sections discuss the available options.

Server Configuration Options

The following server configuration options can be used to monitor client activity:

- Specifying `DIAGNOSTICS LOGON_COMM` in the server configuration file causes the server to write detailed logon status messages to its console. This option is useful for tracking down the cause of failed client connection attempts.
- Specifying `DIAGNOSTICS TRAP_COMM` in the server configuration file causes the server to log all communication messages received from c-tree clients to the file *TRAPCOMM.FCS*. This option provides a way to capture the exact sequence of client requests, and if a copy of the initial data and index files are preserved, the **cttrap** utility can be used to replay this *TRAPCOMM.FCS* log in order to reproduce the original client activity. Note that using this option may impact server performance.
- Specifying `FUNCTION_MONITOR YES` in the server configuration file causes the c-tree Server to display details about each client request to the function monitor window or standard output. Another variation is to specify `FUNCTION_MONITOR <logfile>`, which causes the server to log function details to the file named *<logfile>*. When function monitor output is directed to a log file, the function return codes are included, which makes this option useful for tracking down the occurrence of unexpected c-tree errors. Note that using this option may impact server performance.

ctadmn Utility Options

The c-tree Server Administration Utility, **ctadmn**, can be used to monitor c-tree client activity by following these steps:

1. Start ctadmn. When prompted, enter the user ID of a member of the ADMIN group, the user's password, the optional *FAIRCOM.FCS* file password, and name of the c-tree Server.
2. Select option 4, "Monitor Clients", from the **ctadmn** main menu.
3. Select option 1, "List Attached Clients", from the Monitor Clients menu. **ctadmn** displays an entry such as the following for each connected client:

```
UserID: GUEST                              NodeName:
Task 11                                    Communications: F_TCPIP
         Memory:  25K       Open Files: 2      Logon Time:   0:02
         Tran Time:   0:01  Rqst Time:   0:00  InProcess  Rqst# 13  TRANEND
```

The meaning of each field is shown in the following table:

Field	Explanation
UserID	The user ID specified by this client when connecting to the server.
NodeName	The application-assigned node name for this client.
Task	The server-assigned task ID for the server thread servicing this client. Entries logged by this thread to the server status log include this task ID (for example, "- User 11").
Communications	The communication protocol used by this client.
Memory	The current server memory usage attributed to this client thread.
Open Files	The current number of open files by this client.
Logon Time	The total logon time for this client connection.
Tran Time	The current elapsed transaction time for this client's currently active transaction. Watch for unexpected high transaction times.
Rqst Time	If the text to the right of this value reads "InProcess", this thread is currently executing a request on behalf of a client and this time is the current elapsed time for this client's current request. In this case, a large "Rqst Time" value indicates that the server is taking a long time to complete this client's request. If the text to the right of this value reads "NoRequest", this thread is waiting for the next client request and this time is the current elapsed time since the completion of the previous client request. In this case, a large "Rqst Time" value indicates that it has been a long time since the server has received a request from this client.
Rqst#	If the text to the left of this value reads "InProcess", the function number and function name refer to the c-tree API function currently being executed on behalf of the client. If the text to the left of this value reads "NoRequest", the function number and function name refer to the c-tree API function that were most recently executed on behalf of the client.

SystemConfiguration Options

The c-tree Plus **SystemConfiguration()** API function can be used to monitor c-tree client activity. **SystemConfiguration()** returns three values referenced with the following constants used as subscripts in the output array of LONG values as shown in the following table:

Array Subscript	Explanation
cfgLOGONS	Current number of logons.
cfgUSERS	Maximum number of logons set in server configuration file.
cfgMAX_CONNECT	Maximum number of logons set by server activation.

Monitoring c-tree Server Transaction Activity

This section discusses the available options for monitoring the c-tree Server's transaction activity.

Server Configuration Options

The `CHECKPOINT_MONITOR` server configuration keyword can be used to cause the server to log the starting and completion times of checkpoint operations to the server status log and the server console. This option can be used to determine how often checkpoints occur. If checkpoints occur very

frequently, the server's performance can be affected. In such a situation, the CHECKPOINT_INTERVAL and LOG_SPACE keywords can be used to reduce the frequency of checkpoint operations.

SnapShot API Options

The **SnapShot()** function can be used to retrieve transaction activity details. See the discussion of the *SNAPSHOT.FCS* file format for details.

Monitoring c-tree Server File Usage

This section discusses options for monitoring c-tree data and index file usage by the c-tree Server and its clients.

Server Configuration Options

The following server configuration options support monitoring the c-tree Server's usage of data and index files:

- DIAGNOSTICS LOWL_FILE_IO is useful in troubleshooting file open, create, close, delete, and rename errors. This keyword causes the server to log to the server status log, *CTSTATUS.FCS*, the filename and system error code for failed file open, create, close, delete, and rename operations.
- DIAGNOSTICS WRITETHRU causes the c-tree Server to log to the server status log the names of all *PREIMG* or non-transaction c-tree data and index files that are opened or created during server operation without the *WRITETHRU* filemode in effect.
- DIAGNOSTICS EXTENDED_TRAN_NO causes each physical open of a non-extended-transaction-number file to be listed in *CTSTATUS.FCS*. The reason to check for a file that does not support extended transaction numbers is that if all files do not support extended transaction numbers, then the exceptions could cause the server to terminate if the transaction numbers go out of the original 4-byte range and one of these files are updated. By "all files" we mean superfile hosts and indices; data files are not affected by the extended-transaction-number attribute.

SystemConfiguration API Options

The c-tree Plus **SystemConfiguration()** API function can be used to monitor c-tree data and index file usage by the c-tree Server. **SystemConfiguration()** returns three values referenced with the following constants used as subscripts in the output array of LONG values as shown in the following table:

Array Subscript	Explanation
cfgOPEN_FILES	The number of c-tree Plus files opened by the c tree Server.
cfgPHYSICAL_FILES	The number of physical c-tree Plus files opened by the c-tree Server. Includes superfile members omitted from cfgOPEN_FILES count.
cfgOPEN_FCBS	Number of c-tree Plus file control blocks in use by the c-tree Server.

Monitoring c-tree Server Dynamic Dumps

The c-tree Server logs dynamic dump error messages and error codes to the server status log, *CTSTATUS.FCS*. In the event of a failed dynamic dump, examine the server status log.

The c-tree Server can be configured to log more detailed dynamic dump progress entries to the server status log, including an entry for each file included in the dump, by adding DIAGNOSTICS DYNDUMP_LOG to the server configuration file before starting the server.

Monitoring c-tree Server Automatic Recovery

If `RECOVER_DETAILS YES` is present in the server configuration file when the server is started, the server logs the time spent for each phase of recovery to the server status log. This option provides a way to more specifically monitor the progress of automatic recovery. The server writes the entry for each recovery phase upon completion of that recovery phase, so the current recovery phase can be determined by examining the contents of the server status log.

See the section "Server Startup Hangs or Takes Excessive Time" of this document for more details on monitoring automatic recovery.

Monitoring c-tree Server Cache Usage

The c-tree Plus **SystemConfiguration()** API function can be used to monitor the usage of the c-tree Server data and index caches. **SystemConfiguration()** returns nine values referenced with the following constants used as subscripts in the output array of LONG values an application can use to capture system-wide cache and buffer statistics, (cache pages hold data record images and buffers hold index nodes), allowing an application to track the use of these resources.

Array Subscript	Explanation
cfgCACHE_PAGES	Available cache pages
cfgCACHE_PAGES_INUSE	Cache pages in use
cfgCACHE_PAGES_MXUSE	Maximum cache pages used
cfgCACHE_PAGES_DED	Available dedicated cache pages
cfgCACHE_PAGES_DEDINUSE	Dedicated cache pages in use
cfgCACHE_PAGES_DEDMXUSE	Maximum dedicated cache pages used
cfgBUFFER_PAGES	Available index buffers
cfgBUFFER_PAGES_INUSE	Index buffers in use
cfgBUFFER_PAGES_MXUSE	Maximum index buffers used

Note: The dedicated cache pages are a subset of the regular cache pages.

Monitoring c-tree Server Status Log Messages

During server operation, the c-tree Server writes messages to the server status log, *CTSTATUS.FCS*. The messages the server logs may be informational, warning, or error messages. The system administrator should monitor the server status log to detect situations in which the server writes unexpected warning or error messages to the status log.

The **ctsysm** utility can be used to monitor status log messages. This utility reads a configuration file containing the possible status log messages and associated actions depending on the context of the message. As the server logs messages to the status log, the utility examines the messages and outputs the corresponding message code, which can be matched to the appropriate action, if any, for the message. For details on using the **ctsysm** utility to monitor the c-tree Server status log, see "ctsysm Utility Reference".

Monitoring c-tree Server Process State

The state of the c-tree Server process can be monitored using system utilities and c tree Server utilities. System utilities commonly available on Unix systems include the following:

- ps lists the current processes on the system. This utility can be used to confirm that the server process exists and is in a running state.
- truss traces system calls and signals for a command or process. This utility can be used to determine what operations the server is currently performing.
- pstack prints a hex and symbolic stack trace for each lwp in a process. This utility can be used to determine what each server thread is currently doing. Also see the proc man page for details on other process monitoring tools.
- gcore creates a core image of the specified process. This utility can be used to view a detailed snapshot of the server's operation at a point in time.

The Windows Performance Monitor and Task Manager can be used to observe the state of the c-tree Server process and its current activity.

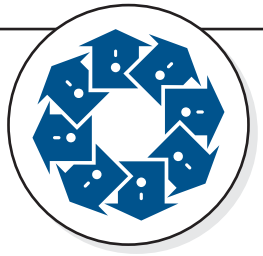
The **ctstat** utility can be used to monitor server activity. By observing c-tree Server statistics over time, the c-tree Server's activity can be measured. For example, transaction commits, communication read/write operations, and disk read and write operations can be examined to determine what the server is currently doing.

The **ctadmn** utility can be used to monitor client activity. Details can be listed for all client connections, including current request, elapsed request time, and elapsed transaction time.

c-tree Server Monitoring Checklist

Below are recommended monitoring procedures for system administrators to follow to ensure proper monitoring of the c-tree Server:

- Use system tools and c-tree Server utilities to monitor the state of the c-tree Server process. The **ctadmn** and **ctstat** utilities can be used for this purpose.
- Use system tools and c-tree Server utilities to monitor the state of c-tree clients and client activity. The **ctadmn** utility and server configuration keywords can be used for this purpose.
- Use system tools and c-tree Server utilities to monitor system and c tree Server resource usage. The **ctstat** utility can be used to collect historical system resource usage and performance statistics in order to detect unexpected changes in resource usage or performance.
- Monitor the c-tree Server status log for unexpected warning and error messages. The **ctsysm** utility can be used for this purpose.
- Monitor c-tree API function return codes. The application should log occurrences of unexpected errors.
- Consider connecting a **ctadmn** client to the server and leaving it connected so that client activity can be monitored even if the server enters a state in which it refuses client connections.



c-tree Server Troubleshooting

This chapter discusses possible c-tree Server problem situations and how to resolve them.

4.1 Failures During c-tree Server Startup

This section discusses failures that may occur when starting the c-tree Server.

Server Fails to Start

There are situations in which an attempt to start the c-tree Server process can fail. A failed server startup can be detected by examining the list of processes running on the system. If ctsrvr (the name of the server binary) is not shown as a running process after the server is started, the server startup has failed.

The **ps** or **pgrep** utilities can be used on Unix systems to list active processes.

The Windows Task Manager can be used on Windows system to list active processes.

In the event of a failed c-tree Server startup, the server logs error messages to *CTSTATUS.FCS* and sometimes to the server console. Check for errors in these locations first in order to understand the reason the server failed to start. If the c-tree Server has successfully started in the past, consider what might have changed since the last successful server startup (such as server configuration file options, server binaries, etc.).

The following are possible causes of a failed server startup:

Unactivated c-tree Server

If the c-tree Server is not pre-activated by FairCom, it must be activated before it is started. If an unactivated server is started, the server writes the following message to *CTSTATUS.FCS* and terminates:

```
Thu Sep 25 15:42:38 2003
- User# 01SERVER NEEDS ACTIVATION KEY!
Execute 'fcactvat' program first
```

To resolve this type of failed startup, activate the c-tree Server using the c tree Server Activation Utility.

Missing or Incorrect Configuration File

The server may fail to start if the server configuration file is missing or contains settings that are inconsistent with those used previously when starting the c-tree Server. For example, if the server configuration file specifies a PAGE_SIZE setting that differs from the setting used when the server

created its *FAIRCOM.FCS* file, the server is unable to open *FAIRCOM.FCS* and server startup fails. In this situation, the server logs error details to *CTSTATUS.FCS*.

To resolve this type of failed startup, review the startup errors logged to *CTSTATUS.FCS* and start the server using a server configuration file with the appropriate configuration settings.

Unrecognized Keyword in Server Configuration File

If the server configuration file used when starting the c-tree Server contains a keyword that is used incorrectly or is not recognized, the c-tree Server fails to start up. The example below shows the messages logged to *CTSTATUS.FCS* when an unrecognized keyword <unrecognized_keyword> is specified in the server configuration file:

```
Thu Sep 25 16:45:06 2003
 - User# 01DO NOT RECOGNIZE CONFIGURATION KEYWORD...: 2
Thu Sep 25 16:45:06 2003
 - User# 01<unrecognized_keyword>: 2
Thu Sep 25 16:45:06 2003
 - User# 01O1 M2 L73 F9 P0x (recur #1) (uerr_cod=0)
```

To resolve this type of failed startup, review the errors logged to *CTSTATUS.FCS* and make the appropriate changes to the server configuration file.

Server Fails to Open Server Administrative Files

The c-tree Server will fail to start if it is unable to open its administrative files, which include the files *FAIRCOM.FCS*, *SYSLOGDT.FCS*, and *SYSLOGIX.FCS*. In this situation, check the server status log and look for a message such as the following:

```
Wed Oct  1 12:34:25 2003
 - User# 01     Could not initialize server. Error: 14
```

If the server fails to start up and logs this message to the server status log, attempt to open *FAIRCOM.FCS*, *SYSLOGDT.FCS*, and *SYSLOGIX.FCS* to determine which of these files failed to open. The files *SYSLOGDT.FCS* and *SYSLOGIX.FCS* are the server's event logs. If the server fails to open the files they can be safely deleted if their contents are not of interest, or they can be rebuilt, or they can be moved out of the server directory and the server will re-create these files during server startup. The file *FAIRCOM.FCS* contains user and group definitions. If the sever fails to open the files and the server's default user and group account settings have not been changed, *FAIRCOM.FCS* can be deleted and the server will re-create it during server startup. If the server administrator has made changes to the user and group accounts settings, rebuild this file or restore it from backup.

If the message shown below appears in the server status log, the server was not able to open *FAIRCOM.FCS* because the current PAGE_SIZE setting differs from the page size used when *FAIRCOM.FCS* was created.

```
Wed Oct 01 12:46:38 2003
 - User# 01     Could not process User/Group Information: 417
```

To correct this problem, change the PAGE_SIZE setting to the correct value, use the **ctscmp** utility to change the page size for *FAIRCOM.FCS*, or delete *FAIRCOM.FCS*.

If the server fails to start due to a failure to open its administrative files, consider adding `DIAGNOSTICS LOWL_FILE_IO` to the server configuration file. This diagnostic option causes the server to log messages showing filenames and system error codes for failed file open/close/delete/create/rename operations to the server status log.

Missing Server Binary or Communication DLLs

The c tree Server can fail to start if the server binary is missing or in the case of the Windows server if a required communication DLL is missing. If the server binary is missing, the system command used to

start the server typically outputs a message indicating that the binary was not found. In the event of a missing communication DLL, *CTSTATUS.FCS* shows a message such as the following:

```
Thu Sep 25 15:15:58 2003
 - User# 01     F_TCPIP: 145
Thu Sep 25 15:15:58 2003
 - User# 01     Could not establish logon area. Error: 143
```

To resolve this type of failed startup, review the startup errors reported by the server startup command or logged to *CTSTATUS.FCS* and copy the necessary binaries to the server's working directory.

Server Cannot Initialize Communication Protocol

If the c-tree Server is unable to initialize its communication subsystem at startup, it will terminate with an error. Examples of this type of failure include a missing communication DLL (as discussed above), improper system configuration for the specified communication protocol, or unavailable communication resources (for example, the TCP/IP port the server is attempting to use is already in use or otherwise unavailable).

To resolve this type of failed startup, review error messages logged to *CTSTATUS.FCS* to determine the reason for the failure. Correct the problem and restart the server.

Missing or Corrupt Server Settings File

Some versions of the c-tree Server are require an encrypted server settings file to exist at startup. A server that requires a settings file will terminate at startup if the settings file is missing or if the server cannot read the settings file. In this situation, the server writes one of the following messages to *CTSTATUS.FCS*:

```
Thu Sep 25 16:41:10 2003
- User# 01     The Server's settings file is missing.
It is required to operate this server.
Thu Sep 25 16:42:01 2003
- User# 01     The current Server's settings file is invalid.
Use current FairCom utility to recreate the settings file.
```

To resolve this type of failed startup, review error messages logged to *CTSTATUS.FCS*. Make available to the server a good copy of the encrypted settings file and restart the server.

Automatic Recovery Fails

Each time the c-tree Server starts, it examines its transaction logs to determine whether or not it needs to perform automatic recovery of *TRNLOG* files. The server automatically performs automatic recovery if it determines automatic recovery is required. When automatic recovery is successful, the server continues its normal startup processing. If automatic recovery fails, the c-tree Server logs an error message to *CTSTATUS.FCS* and terminates.

To resolve this type of failed startup, review the error messages logged to *CTSTATUS.FCS* and take the appropriate action. See the "Automatic Recovery Fails" section in this chapter for a discussion of specific types of automatic recovery failures and what to do in each case.

A Server is Already Running in the Working Directory

Two c-tree Servers cannot be running in the same working directory at the same time.

> **Note:** Here the working directory refers to the directory in which the server stores its transaction logs and other **.FCS* files.

If a server attempts to start in a directory in which another server is already running, the server fails to start up and logs the following error message to *CTSTATUS.FCS*:

```
Thu Sep 25 16:38:15 2003
 - User# 01     Is another server running in this workspace?
Thu Sep 25 16:38:15 2003
 - User# 01     O1 M99 L54 F537 P1x (recur #1) (uerr_cod=0)
```

To resolve this type of failed startup, shut down the running server or configure the c-tree Server to start in a different working directory, then start the c-tree Server.

Dynamic Dump Cannot Be Scheduled

If the server configuration file includes the DUMP keyword, the server opens the specified dynamic dump script file and attempts to schedule a dynamic dump. If the dump cannot be scheduled (for example, due to a missing dump script or a dump script with incorrect or unrecognized keywords), the server logs an error message to *CTSTATUS.FCS* and shuts down. Below is an example showing errors logged to *CTSTATUS.FCS* when a dynamic dump cannot be scheduled at server startup because the dump script file does not exist:

```
Thu Sep 25 16:48:43 2003
 - User# 01     DD: could not open script file...: 12
Thu Sep 25 16:48:43 2003
 - User# 01     my.scr: 12
Thu Sep 25 16:48:43 2003
 - User# 01     Could not schedule Dynamic Dump...: 5
Thu Sep 25 16:48:43 2003
 - User# 01     my.scr: 5
```

To resolve this type of failed startup, review the error messages logged to *CTSTATUS.FCS* to understand the specific cause of the failure. Correct the problem that prevented the scheduling of the dynamic dump and restart the server.

Server Startup Terminates Abnormally

In addition to the specific causes of a failed server startup, the c-tree Server may terminate abnormally at startup for the following reasons:

- The c-tree Server process encounters a fatal exception, causing the system to terminate the server process. In this case the system may produce a core image of the server process at the time of the exception. The c-tree Server status log may contain error messages related to the exception.
- An administrator forcibly terminates the c-tree Server process. In this case, the process is abruptly terminated and the c-tree Server status log does not show an indication of a shutdown occurring.
- The system on which the c-tree Server is running terminates abnormally (due to power loss, operating system exception, or sudden system reboot). In this case, the server process may be abruptly terminated, which case the status log does not show an indication of a shutdown occurring.
- The c-tree Server detects an unexpected internal server error situation known as a *catend* or a *terr*. In these cases, the server status log shows an error message containing details about the internal server error.

If a server startup terminates abnormally, follow these steps:

1. Examine system logs, application logs, and the server status log to determine the nature of the abnormal server termination.
   a) If a fatal exception terminated the server process, save the core file if it exists.
   b) If the server terminated due to a fatal exception or internal c-tree Server error, save a copy of the server's **.FCS* files, the server configuration file, and if time and disk space permit save a copy of all data and index files before restarting the c-tree Server. These files can be used to analyze the abnormal server termination.
   c) Consider whether any recent hardware or software changes could explain the reason for the startup failure (including server configuration file option changes).

d) If the situation that led to the abnormal server termination can be understood by analyzing the server status log or other system logs, correct the problem that caused the server to terminate. For example, if the server terminated due to insufficient disk space which prevented the server from writing to its transaction logs, free up disk space to ensure the server has enough space for the transaction logs (Caution - but do not delete active transaction logs before the server performs its automatic recovery).

2. Unlike an abnormal server termination that occurs after the server is operational, an abnormal server termination at server startup does not require special actions to be taken to ensure the integrity of *PREIMG* and non-transaction c-tree data and index files because the application data and index files will not have been open with active changes.
3. If the abnormal server termination occurred during automatic recovery, the *TRNLOG* files are in an unknown state. Automatic recovery must be successfully completed or the *TRNLOG* files re-created or restored from backup. See the “Automatic Recovery Fails” topic in the “Failures During System Recovery” section of this document for details on the steps to follow if the server terminates abnormally during automatic recovery.
4. If the server can be restarted and automatic recovery completes successfully, clients can connect to the server and can resume processing.

Server Startup Hangs or Takes Excessive Time

The c-tree Server startup process is usually very fast. The c-tree Server logs a message of the form shown below to *CTSTATUS.FCS* when startup is complete and the server is ready for clients to connect:

```
c-tree(R) V8.13.826 Server Is Operational      -SN 33333333
```

If the c-tree Server process appears in the system process list but *CTSTATUS.FCS* does not yet show the above message indicating the server is operational, the most likely cause is that the server is performing automatic recovery and for some reason the recovery is taking a long time.

To determine if the long startup time is due to automatic recovery occurring, check the server status log. When the server begins automatic recovery, it writes the following message to the status log:

```
- User# 01     Beginning automatic recovery process
```

When the server completes automatic recovery, it writes the following message to the status log:

```
- User# 01     Automatic recovery completed
```

If the server has logged the first message to the status log but not the second, it has begun but has not yet completed automatic recovery.

If `RECOVER_DETAILS YES` is present in the server configuration file when the server is started, the server logs the time spent for each phase of recovery to the server status log. This option provides a way to more specifically monitor the progress of automatic recovery. Below is an example showing the order of automatic recovery phases. The server writes the entry for each recovery phase upon completion of that recovery phase, so the current recovery phase can be determined by examining the contents of the server status log.

```
 Mon Sep 29 10:14:34 2003
 - User# 01     Transaction scan time:    1 seconds.
Mon Sep 29 10:14:34 2003
 - User# 01     Beginning automatic recovery process
Mon Sep 29 10:14:34 2003
 - User# 01     Index repair time:        0 seconds.
Mon Sep 29 10:14:34 2003
 - User# 01     Index composition time:   0 seconds.
Mon Sep 29 10:14:34 2003
 - User# 01     Transaction undo time:    0 seconds for   2 transactions.
Mon Sep 29 10:14:34 2003
 - User# 01     Vulnerable data time:     0 seconds.
Mon Sep 29 10:14:34 2003
 - User# 01     Vulnerable index time:    0 seconds.
```

```
Mon Sep 29 10:14:34 2003
 - User# 01     Transaction redo time:    0 seconds for 843 transactions.
Mon Sep 29 10:14:34 2003
 - User# 01     Automatic recovery completed
```

Although the current recovery phase and time spent so far during recovery can be determined using the above approach, the log entries do not indicate how much time remains until recovery is complete.

If automatic recovery appears to hang or is taking a long time, the two options are to wait until recovery completes or to abandon recovery and restore *TRNLOG* data and index files from a backup. See the "Automatic Recovery Fails" topic in the “Failures During System Recovery” section of this document for details on these options.

If the server startup is taking a long time or appears to hang and the cause does not appear to be automatic recovery (based on the status log messages), check the state of the server process using system utilities as described in the “Monitoring c-tree Server Process State” section. If necessary, the server can be forcibly terminated and restarted without affecting the state of c-tree data and index files because the files will not have been open for end user modification at this point.

4.2 Failures During c-tree Server Operation

This section discusses failures that may occur during c-tree Server operation following a successful startup.

Clients Cannot Connect to Server

To connect to the c-tree Server, a client calls a c-tree Plus API function such as **InitCtree()**, **InitCtreeXtd()**, **InitISAM()**, **InitISAMXtd()**, or **CTDBLogon()**. A connection attempt may fail for various reasons. Check the function return code to determine possible causes for the connection failure. The following table sections lists errors a c-tree Plus API function may return in the event of a failed connection attempt and describes possible causes and troubleshooting steps for each:

c-tree Error 10: SPAC_ERR

Error description

Memory allocation error during logon.

Possible causes

An attempt to allocate memory during logon failed. The most common cause is a shortage of system memory.

Troubleshooting steps

Check system memory usage by the c-tree Server and other processes on the system. Free system memory (for example by stopping unnecessary processes or shutting down and restarting the c-tree Server) and attempt to logon again.

c-tree Error 84: MUSR_ERR

Error description

Maximum users exceeded.

Possible causes

The c-tree Server enforces a limit on the number of concurrently connected clients. This error indicates that the maximum number of concurrent connections has been reached.

Troubleshooting steps

In some cases, it is expected that the maximum number of clients may be logged on. In this situation, try the operation again after clients have logged off. If reaching the connected user limit is unexpected, use the **ctadmn** utility to list the current client connections and to view their activity. Use **ctadmn** to determine if there are any inactive client connections that can be terminated. When the number of connected clients is below the connection limit, try to logon again. Note, the Server is designed to allow one instance of the user ID “ADMIN” to be able to connect to the Server even if the maximum number of supported connections has been reached.

Note: A client which belongs to the ADMIN group and which sets the *USERPRF_ADMSPCL* bit of the user profile can log onto to the c-tree Server even if the server already has the maximum permitted number of clients logged on. This capability is useful in situations in which it is necessary to perform system administration even when the client limit has been reached. The **ctadmn** utility automatically uses this feature.

c-tree Error 127: ARQS_ERR

Error description:
Could not send request. A communication error occurred while sending a request to the server.

Possible causes and troubleshooting steps:
The section "Clients Lose Connection to Server" describes possible causes and troubleshooting steps for this error.

c-tree Error 128: ARSP_ERR

Error description:
Could not receive answer. A communication error occurred while receiving a response from the server.

Possible causes and troubleshooting steps:
The section "Clients Lose Connection to Server" describes possible causes and troubleshooting steps for this error.

c-tree Error 133: ASKY_ERR

Error description:
The server could not be located.

Possible causes:

1. The c-tree Server is not operational.
2. The specified server name or location is incorrect.
3. Network connectivity problems prevent the client from connecting to the server.
4. The client is using a different communication protocol than the server is using.

Troubleshooting steps:

1. Verify that the c-tree Server is operational. Use system utilities to confirm that the server process is active.

2. Confirm that the server is using the same communication protocol as the client. This can be determined by examining the server configuration file (*ctsrvr.cfg*) and the server status log (*CTSTATUS.FCS*).
3. Verify that the server name and location are correctly specified for the communication protocol the client is using. Note the SERVER_NAME value is case sensitive. If the TCP/IP communication protocol is being used, be sure the Machine_Name@Server_NameServer_Name@Machine_Name protocol is being followed. For example, 128.128.128.128@FAIRCOMSFAIRCOMS@128.128.128.128.
4. Check the network connectivity. If the server and client reside on separate machines, ping the server machine from the client machine using the host name or IP address specified when connecting. Use system utilities to verify that the server is listening for incoming connections. Try connecting using ctadmn from the server machine and from the client machine.
5. Add the option DIAGNOSTICS LOGON_COMM to the c-tree Server configuration file and restarting the server. This option causes the server to write detailed logon status messages to its console. These messages can help determine at what point the connection attempt failed. For example, the messages can show whether or not the server was aware of the client's connection attempt. Note: tracking of logon details on the client side can be enabled by compiling the c-tree client library with #define CT_DIAGNOSE_LOGON_COMM enabled. The client outputs logon messages to standard output.
6. Shut down and restart the c-tree Server to cause it to reinitialize its communication subsystem.

c-tree Error 150: SHUT_ERR

Error description:
Server is shutting down. The client connected to the server but the server closed the connection.

Possible causes and troubleshooting steps:
The section "Clients Lose Connection to Server" describes possible causes and troubleshooting steps for this error.

c-tree Error 162: SGON_ERR

Error description:
Server has gone away. The client connected to the server but the server closed the connection.

Possible causes and troubleshooting steps:
The section "Clients Lose Connection to Server" describes possible causes and troubleshooting steps for this error.

c-tree Error 450: LUID_ERR

Error description:
Invalid user ID.

Possible causes:
The specified user ID does not exist.

Troubleshooting steps:

1. Use the **ctadmn** utility to list the user IDs recognized by the server.
2. Logon using a valid user ID or create a new user account with the specified user ID.

c-tree Error 451: LPWD_ERR

Error description:
Invalid password.

Possible causes:
The specified password is invalid for the specified user ID. Note that passwords are case-sensitive.

Troubleshooting steps:
To resolve this error, logon using the correct password for the specified user ID or use the **ctadmn** utility to change the password to match the specified password.

c-tree Error 452: LSRV_ERR

Error description:
Server could not process user or account information. The c-tree Server encountered an error when reading the account information for the specified user.

Possible causes:
This logon error points to possible problems with the user account data in the *FAIRCOM.FCS* superfile. Depending on the location of the problem, the c tree Server may log one of the following messages to *CTSTATUS.FCS*:

```
User ID file processing error
User/Group file processing error
Valid ID file processing error
```

Troubleshooting steps:
Use the **ctadmn** utility to list the properties for the specified user account. If necessary, change account properties or delete and re-create the account. If the error persists, using the **ctscmp** utility to compact *FAIRCOM.FCS* may help correct this type of error. Or, if a good backup copy of *FAIRCOM.FCS* exists, restoring this file from backup is an option.

c-tree Error 470: LGST_ERR

Error description:
Guest logons disabled.

Possible causes:
The client attempted to logon as guest by specifying a NULL or empty user ID, but guest logons are disabled. When GUEST_LOGON NO is specified in a server settings or configuration file, the guest account is disabled.

Troubleshooting steps:
To avoid this error, logon using a non-guest account or reconfigure the c-tree Server to allow guest access.

c-tree Error 530: LMTC_ERR

Error description:
Client does not match server.

Possible causes:
Some c-tree Servers only accept connections by specially-configured c-tree clients. When a standard c-tree client attempts to connect to such a server, the server rejects the connection attempt with c-tree error 530.

Troubleshooting steps:
To avoid this error, connect to the server using the appropriate specially-configured c-tree client.

c-tree Error 579: LIVL_ERR

Error description:
Logon interval error. The c-tree Server can be configured to require a user to logon at least once within a defined time. This ability can be configured on a per-user account basis and on a system-wide basis if desired. If the user fails to logon at least once within the prescribed time period, the c-tree Server disables the user account and subsequent logon attempts using that user ID fail with c-tree error 579.

Possible causes:
A required logon time period is in effect for the specified user account and the user did not logon at least once within the specified time period.

Troubleshooting steps:
An administrator can re-enable a user account that the c-tree Server has deactivated due to exceeding the required logon interval by using the **ctadmn** utility.

c-tree Error 584: LRSM_ERR

Error description:
Exceeded failed logon limit. The c-tree Server supports setting a limit on the number of consecutive logon failures due to specifying an invalid password. This limit can be set on a per-user and on a system-wide basis. If a user exceeds the specified number of consecutive logon failures, the server disables the user account and subsequent logon attempts using that user ID fail with c-tree error 584.

Possible causes:
A consecutive logon failure limit is in effect for the specified user account and the user has exceeded the limit.

Troubleshooting steps:
An administrator can re-enable a user account that the c-tree Server has deactivated due to exceeding the failed logon limit by using the **ctadmn** utility.

c-tree Error 585: LVAL_ERR

Error description:
Logon date exception. The c-tree Server supports setting starting and ending validity dates for user accounts. An attempt to logon using a user account before its starting validity date or after its ending validity date fails with c tree error 585.

Possible causes:
A validity period is in effect for the specified user account and the starting validity date has not yet arrived, or the ending validity date has already passed.

Troubleshooting steps:
An administrator can change the starting and ending validity dates for a user account using the **ctadmn** utility.

c-tree Error 589: LADM_ERR

Error description:
Member of ADMIN group required.

Possible causes:
A client is attempting to connect to the c-tree Server in an administrative mode (for example, logging on using **ctadmn** or logging on in order to shut down the server) but the specified user ID is not a member of the ADMIN group.

Troubleshooting steps:
Logon using a user ID that is a member of the ADMIN group, or use the **ctadmn** utility to add the specified user ID to the ADMIN group.

c-tree Error 593: XUSR_ERR

Error description:
Non-ADMIN user blocked from logon. The c-tree Server can be put into an operational mode in which only users that are members of the ADMIN group are allowed to logon. When this mode is in effect, logon attempts by non-ADMIN users are rejected with c-tree error 593.

Possible causes:
The server administrator has enabled the c-tree Server's ADMIN group only logon mode using the c-tree Plus SECURITY API function or using the STARTUP_BLOCK_LOGONS keyword in the server configuration file.

Troubleshooting steps:
To resume normal server operation, the server administrator can call the c tree Plus SECURITY API function with a mode of SEC_BLOCK_OFF.

c-tree Error 609: LTPW_ERR

Error description:
One-use temporary password failure. The c-tree Server supports establishing a one-time password that another client can use (along with the user ID matching the caller that set the password) as long as the original caller is still logged on. Once the original caller logs off, the one-time password becomes invalid. Once the password is used by the subsequent client, the password is no longer available.

Possible causes:
Either no one-time password is in effect for the specified user ID, or the specified password is invalid.

Troubleshooting steps:
Confirm that the client that established the one-time password is already logged on, that a client has not yet connected using the one-time password, and that the specified password is correct.

Clients Lose Connection to Server

A connected client may lose its connection to the c-tree Server for various reasons. If a client loses its connection to the server, subsequent calls to c-tree functions that send requests to the server return an error indicating that the connection has been lost. The following table sections lists errors a c-tree Plus API function may return in the event of a lost connection and describes possible causes and troubleshooting steps for each:

c-tree Error 7: TUSR_ERR

Error description:
Terminate user. The c-tree Server or server administrator has terminated the client's connection to the server.

Possible causes:

1. An administrator may have terminated the client connection.
2. The c-tree Server may have terminated the client connection (for example, when the server is shutting down or when the server aborts a transaction).

Troubleshooting steps:
Examine *CTSTATUS.FCS* to determine the reason the connection was terminated. See the discussion of c-tree error 127 for additional details about reconnecting to the c-tree Server after a client connection is terminated.

c-tree Error 127: ARQS_ERR

Error description:
Could not send request. A communication error occurred while sending a request to the server.

Possible causes:

1. The server may have shut down.
2. The server may have terminated abnormally.
3. An administrator may have terminated the client connection.
4. A network error may have occurred that terminated the connection to the server.

Troubleshooting steps:

1. Verify that the server is operational. Restart the server if it is not operational.
2. When a client loses its connection to the c-tree Server and the server detects the lost connection, the server aborts the client's active transaction (if any) and closes any files that the client had open. If the server is still operational, use the **ctadmn** utility to confirm that the server thread for the lost connection has been terminated. If necessary, terminate the old connection using **ctadmn**.
3. To determine if the connection was terminated by an administrator, check *CTSTATUS.FCS* for messages of the form:

```
External kill request posted against user #<taskID>
```

where <taskID> is a server-assigned thread task ID.

1. Attempt to reconnect to the server.
2. If this communication error occurs at logon time, consider using the DIAGNOSTICS LOGON_COMM server configuration keyword to monitor logon process as described in the discussion of c-tree error 133.
3. If this communication error occurs frequently without explanation (for example, connections are lost but the c-tree Server remains active), investigate the possibility of network connectivity errors.

c-tree Error 128: ARSP_ERR

Error description:
Could not receive answer. A communication error occurred while receiving a response from the server.

Possible causes and troubleshooting steps:
The possible causes and troubleshooting steps are the same as for error 127.

c-tree Error 150: SHUT_ERR

Error description:
Server is shutting down. The client connected to the server but the server closed the connection.

Possible causes:
The c-tree Server is refusing new connections because it is in the process of shutting down.

Troubleshooting steps:
Check the status of the c-tree Server process. Restart the c tree Server if necessary. When the server is operational, retry the connection attempt.

c-tree Error 162: SGON_ERR

Error description:
Server has gone away. The client connected to the server but the server closed the connection.

Possible causes and troubleshooting steps:
The possible causes and troubleshooting steps for this error are the same as described for c-tree error 150.

Number of Active Transaction Logs Unexpectedly Increases

The number of active transaction log files required to support c-tree Server operation is nominally normally 4. Each time the server creates a new log, it determines whether or not the oldest existing log can be deleted (or made inactive, if the `KEEP_LOGS` server configuration option is specified in the server configuration file). A number of conditions require the server to keep more than 4 active log files. It is important for the server administrator to detect cases in which the number of active logs increases significantly and to understand the cause and whether or not action is required.

When the number of log files is about to increase, the server logs the following message to *CTSTATUS.FCS*:

```
The number of active log files increased to <nlogs>
```

where *<nlogs>* is the new number of active logs.

The most important of the situations that require keeping the oldest transaction log active are:

- Increasing CHECKPOINT_FLUSH to delay flushing of buffers associated with committed transactions:

When creating a new transaction log, the c-tree Server determines the recoverability vulnerability due to unflushed buffers associated with committed transactions and keeps the required number of active logs. The following formula can be used to estimate the number of logs required to support unflushed buffer/cache pages based on server configuration settings.

Let:

```
CPF = CHECKPOINT_FLUSH value (defaults to 2)
CPL =  number of checkpoints per log (typically 3 and no less than 3)
MNL =  minimum number of logs to support unflushed pages
```

Then:

```
MNL =  ((CPF + CPL - 1) / CPL) + 2, where integer division is used
```

For example:

```
CPF=2,   CPL=3  => MNL = 3 (But the server enforces a minimum of 4)
CPF=7,   CPL=3  => MNL = 5
CPF=9,   CPL=3  => MNL = 5
CPF=10, CPL=3  => MNL = 6
```

- A pending transaction that began several logs ago and still has not committed or aborted:

Unlike `CHECKPOINT_FLUSH`, which leads to a well-defined limited increase in the number of active transaction logs, a long uncommitted transaction can lead to an unlimited increase in transaction logs. For example, if a client begins a transaction and then sits idle while other clients execute transactions, the other clients' transaction activity fills the transaction logs. When the server creates new logs it finds that the idle client's transaction has not committed. The server must keep the log in which the idle client's transaction begin is logged until that client aborts or commits its transaction. For this reason, it is important to monitor the number of active transaction logs. In the event of an unexpected long

transaction, the **ctadmn** utility can be used to list connected clients and their transaction times and to terminate clients as needed.

- Dynamic dumps:

A dynamic dump is similar to the case of a long pending transaction. The dynamic dump must keep all transaction logs from the dump start time to the dump end time in order to include in the dump stream file all transaction activity that occurred during the dump. If very large files are included in the dump, the dynamic dump can take a significant amount of time. Depending on the amount of transaction activity between the dump start and end times, the number of active logs that must be kept during a dynamic dump can be large.

The c-tree Server logs to *CTSTATUS.FCS* an explanation as to the condition that triggered the increase. When the increase is caused by a pending transaction, the server attempts to identify the user ID and node name associated with the transaction.

Based on the cause of the increased number of active logs shown in *CTSTATUS.FCS*, the server administrator can take the appropriate action, if any. For example, a long pending transaction can be aborted using the **ctadmn** utility to terminate the client that began the transaction, or a dynamic dump can be terminated using **ctadmn**.

Server Is In A Non-Responsive State

The symptoms of a c-tree Server in a non-responsive state are the following:

- Requests from connected clients hang indefinitely.
- Client connection attempts hang or fail with an error such as c-tree error 133.

A non-responsive server can be detected by monitoring connected clients and looking for client requests that have not completed within a reasonable timeframe. Note that some c tree API function calls can be expected to take a significant amount of time (for example, when rebuilding a large file or physically closing a file that has many unwritten updated cache buffers). Even reading a record can take awhile if the call must wait to acquire a lock on the record. The specific symptoms that indicate a fully non-responsive c-tree Server are that all client requests hang and new connection attempts may fail.

When the c-tree Server is in a non-responsive state follow these steps to identify the cause and to correct the problem:

1. The **ctadmn** utility can be used to monitor the status of client connections, but in the event of a non-responsive server, **ctadmn** may be unable to connect to the server or its requests may also hang. If **ctadmn** can connect and can list connected clients, it may be possible to use it to terminate c-tree clients or to shut down the c-tree Server.
2. If **ctadmn** cannot connect to the server, use system monitoring utilities to determine the state of the c-tree Server process. Verify that the process is in a running state and if possible use system utilities to save a core image of the c-tree Server process and stack traces for all the server threads. See Monitoring the c-tree Server and Its Data for details about system utilities that can be used to collect information about the state of the c-tree Server process. Any information that can be collected about the state of a non-responsive server can be useful in determining the cause and finding a way to prevent future occurrences of such a problem.
3. To shut down a non-responsive c-tree Server, follow the steps described in the section "Shutting Down the c-tree Server". Shutting down a non-responsive server might require a hard kill.

Some Clients Are In A Non-Responsive State

A c-tree client can be in a non-responsive state if a server request hangs or takes a long time to complete. As discussed above, some c tree calls (such as rebuild calls or file close calls that involve

physically closing a file) may take awhile to complete. Calls to read a record with a blocking lock will hang until the lock can be acquired.

If requests made by one or more clients do not complete in a reasonable timeframe, follow these steps to identify the cause and to correct the problem:

1. The **ctadmn** utility can be used to view the current state of the client connections, including the function name for the current request being processed on behalf of each client and the current request time.
2. The c-tree Server's snapshot and lock dump capabilities can be used to collect details about the state of the c-tree Server. For example, if **ctadmn** shows one or more clients hanging on record read operations, use the c-tree Plus API function **LockDump()** to dump the current state of the server's lock table to disk, and examine this log to determine if the read requests are simply blocking waiting to acquire a record lock. If this is the case, identify from the log which client currently holds the lock and use **ctadmn** to view the activity of that client or to terminate that client if appropriate.
3. If the cause of the long request time cannot be determined using c-tree Server utilities or c-tree API functions, use system utilities to monitor the server's system calls and to collect details about the server process state. Saving a core image and stack traces for the server threads in this type of situation can help identify the cause of the hanging client requests so that the problem can be resolved.
4. If necessary, use **ctadmn** to terminate client connections that are non-responsive. If after terminating the client connections using **ctadmn**, the client connections still appear in the list of connected clients, the c-tree Server may have to be shut down and restarted to clear the hung connections.

Errors Occur When Opening c-tree Files

A c-tree data or index file can fail to open for a variety of reasons. This section introduces a server configuration keyword that can be used to log system error details in the event of failed file open, create, close, delete and rename operations. The remainder of the section focuses on specific c-tree errors returned by c-tree file open API functions and ways to resolve the errors.

Enabling Low-Level File I/O Diagnostics

The c-tree Server configuration keyword `DIAGNOSTICS LOWL_FILE_IO` is useful in troubleshooting file open, create, close, delete, and rename errors. This keyword causes the server to log to the server status log, *CTSTATUS.FCS*, the filename and system error code for failed file open, create, close, delete, and rename operations. Although client applications have access to system errors through the c tree global variable *sysiocod*, it can be useful to have the server log these errors. For example: An end-user has problems opening a file. The end-user copied the data file from a CD-ROM to the hard disk leaving the file marked read-only. When the user attempted to open these files, the open failed with c-tree error 12. Adding the `DIAGNOSTICS LOWL_FILE_IO` keyword directed the Server to log the system error code to *CTSTATUS.FCS* which helped identify that the file open failed because the file was marked read-only.

c-tree Error 12: FNOP_ERR

Error description:
Could not open file: not there or locked.

Possible causes:

1. The specified filename or path is incorrect (no file by that name exists).

2. The specified file is already open using a file access mode that conflicts with the specified access mode. For example, if the file is open in EXCLUSIVE mode, attempting to open the file in SHARED mode fails (and vice-versa).
3. The server does not have the appropriate permission to access the file (for example the file is marked read-only or system file security attributes are set to disallow access by the user and group under which the server is running).

Troubleshooting steps:

1. Verify that the specified filename and path are correct. Note that filenames on some operating systems are case-sensitive. If using ISAM open functions, it is possible that the data file open succeeded but an index file open failed. Check the isam_fil global variable to determine which file open failed. Try opening the data and index files individually using low-level functions to determine which open fails.
2. Check the c-tree global variable sysiocod. If it is set to -8 (FCNF_COD), this indicates that the open failed due to conflicting access modes.
3. Check the file permissions to verify that the c-tree Server has the appropriate file access permissions (read/write access is usually what is required).
4. If the failed open occurs on a superfile member, verify that the member exists using the c-tree Plus API function **GetSuperFileNames()** and that the member name is specified exactly as it appears in the superfile directory index (member names are always case-sensitive).

c-tree Error 14: FCRP_ERR

Error description:
File corrupt at open.

Possible causes:
The c-tree Server sets an update flag in the header of c-tree data and index files on the first update to the file after the file is opened. The server resets the update flag when the file is physically closed (after all updated cache pages for the file are written to disk). When the server finds the update flag still set when opening the file, the server considers this to mean that the file was updated but was not properly closed, and so the state of the file is unknown. For example, if a file is updated and then the server terminates abnormally, the update flag remains set, which indicates that unwritten updates might not have been written to the file. Error 14 is most likely to occur for PREIMG and non-transaction files. Because the server's automatic recovery processes TRNLOG files in the event of an abnormal server termination, error 14 is not expected for TRNLOG files unless automatic recovery fails. See the discussion of TRNLOG, PREIMG, and non-transaction files for full details about caching and the state of files in the event of an abnormal server termination.

Troubleshooting steps:

1. If the file is a TRNLOG file, review the sequence of events that led up to the error 14. If the c-tree Server terminated abnormally, restarting the server should cause automatic recovery to occur, restoring the TRNLOG files to a consistent transaction state and avoiding the possibility of error 14 occurring.
2. If the file is a PREIMG or non-transaction file and the server terminated abnormally, error 14 can be avoided by rebuilding the affected files, or re-creating the files and re-loading their data from an external source, or by restoring backup copies of the files. To avoid data loss and error 14 for non-TRNLOG files in the event of an abnormal server termination, consider using the WRITETHRU filemode and WRITETHRU server configuration options. See the discussion of the WRITETHRU filemode for details.

c-tree Error 417: SPAG_ERR

Error description:
Cache page size error.

Possible causes:
When a superfile is opened, the index node size currently in effect must be the same as the index node size at the time the superfile was created. This is not usually a problem unless one wishes to move the file between different environments. Error 417 results if the node size does not match. By contrast, an ordinary index file can be opened as long as the current node size is at least as large as the node size at the time the index file was created.

Troubleshooting steps:
To resolve this error, either:

1. Re-create the superfile using the page size setting currently used by the c-tree Server, or
2. Change the server's PAGE_SIZE setting to ensure it matches the page size used when creating the superfile and restart the c-tree Server.

c-tree Error 456: SACS_ERR

Error description:
Group access denied.

Possible causes:
The user that is attempting to open a file is not a member of a group with access to the file.

Troubleshooting steps:
Use **ctadmn** to list the file permissions assigned to the file. To avoid this error, either add the user to a group that has permission to access the file or change the file permissions to allow the user to access the file.

c-tree Error 457: SPWD_ERR

Error description:
File password invalid.

Possible causes:
A password is assigned to the file and the file password specified when opening the file is incorrect.

Troubleshooting steps:
Specify the correct file password for the file, or use the **ctadmn** utility to change or reset the file's password.

Errors Occur When Reading or Writing c-tree Files

c-tree Error 35: SEEK_ERR

Error description:
Seek error. A file seek operation on a c-tree data or index file failed.

Possible causes:
A file seek operation can fail for various reasons, including disk media errors or an invalid file descriptor.

Troubleshooting steps:
In the event of a SEEK_ERR, the c-tree Server logs the following message to *CTSTATUS.FCS*:

```
SEEK_ERR...
```

<filename>

where <filename> is the name of c-tree data or index file for which the seek operation failed.

When a c-tree Plus API function returns c-tree error 35, check the value of the c-tree global variable *sysiocod*, which contains the system error code returned by the failed seek operation. The interpretation of the system error code can explain the cause of the failed seek operation.

c-tree Error 36: READ_ERR

Error description:
Read error. A file read operation on a c-tree data or index file failed.

Possible causes:
A file read operation can fail for various reasons, including disk media errors and inaccessible files due to locking by external applications.

Troubleshooting steps:
When a c-tree Plus API function returns c-tree error 36, check the value of the c-tree global variable sysiocod, which contains the system error code returned by the failed file read operation. The interpretation of the system error code can explain the cause of the failed read operation.

c-tree Error 37: WRITE_ERR

Error description:
Write error. A file write operation on a c-tree data or index file failed.

Possible causes:
A file write operation can fail for various reasons, including disk media errors, insufficient disk space, and inaccessible files due to locking by external applications.

Troubleshooting steps:
When a c-tree Plus API function returns c-tree error 37, check the value of the c-tree global variable sysiocod, which contains the system error code returned by the failed file write operation. The interpretation of the system error code can explain the cause of the failed write operation.

c-tree Error 40: KSIZ_ERR

Error description:
Index node size too large. The c-tree Server was not able to open the specified index file, superfile, or variable-length data file because the page size used when creating the file is larger than the server's current page size. The page size determines the maximum supported index node size.

Possible causes:
The file was created using a larger PAGE_SIZE setting than the server is currently using. This situation can arise if the file was created by a server or a standalone c-tree utility that is configured to use a smaller page size than the server is currently using, or if after the file was created the PAGE_SIZE setting was changed and the server was restarted.

Troubleshooting steps:
To resolve this error, either:

1. Re-create the file (by rebuilding or compacting the file) using the page size setting currently used by the c-tree Server, or
2. Change the server's PAGE_SIZE setting to ensure it is at least as large as the page size used when creating the file and restart the c-tree Server.

Note: A c-tree superfile has stricter page size requirements than a c-tree index has. A superfile can

only be opened by a server whosewho's PAGE_SIZE exactly matches the page size used when creating the superfile. See the discussion of c-tree error 417 for details.

c-tree Error 49: FSAV_ERR

Error description:
Could not save file. In some situations, the c-tree Server must ensure that all writes that have been issued to the filesystem for a c-tree data or index file have been flushed to disk. The server accomplishes this by issuing a "save" operation on the file, which involves a system call that forces the filesystem to write to disk all unwritten filesystem buffers for the file. If this flush fails, the server returns c-tree error 49.

Possible causes:
A file flush operation can fail for a variety of reasons such as a disk media error, insufficient disk space, or a loss of connectivity to network storage system.

Troubleshooting steps:
When a save operation fails, the c-tree Server logs the following message to *CTSTATUS.FCS*:

```
ctsave failed: system code = <err>  lc = <loc>  fd = <fd>
<filename>
```

where <err> is the system error code returned by the failed flush call, <loc> is a c tree Server location code, and <filedesc> is the file descriptor passed to the flush call. Check the interpretation of the system error code to determine the reason why the flush call failed.

c-tree API Call Fails With Unexpected Error

If a c-tree API function call fails with an error that is not described in this document and the reason for the error is not clear from the context of the situation, consult the *c-tree Plus Function Reference Guide* entry for the c-tree Plus API function that returned the error.

Server Writes Unexpected Messages to Status Log

During server operation, the c-tree Server writes messages to the server status log, *CTSTATUS.FCS*. The messages the server logs may be informational, warning, or error messages. The system administrator should monitor the server status log in order to detect situations in which the server writes unexpected warning or error messages to the status log.

The **ctsysm** utility can be used to monitor status log messages. This utility reads a configuration file containing the possible status log messages and associated actions depending on the context of the message. As the server logs messages to the status log, the utility examines the messages and outputs the corresponding message code, which can be matched to the appropriate action, if any, for the message. For details on using the **ctsysm** utility to monitor the c-tree Server status log, see the appendix titled "ctsysm Utility Reference"..

Server Exhibits Atypical Performance

During server operation, the server administrator can use c-tree Server and system monitoring tools to measure performance properties of the c-tree Server. The application may also provide tools used to monitor the performance of the system or the database components of the system.

When these system monitoring utilities detect unexpected performance characteristics of the database components of the system, follow these steps to identify the cause of the unexpected performance so the problem can be understood and resolved:

1. Identify the nature of the unexpected performance characteristics of the system as specifically as possible. For example:

   a) Can specific application operations or c-tree function calls made by clients be identified that are performing differently than expected? Application-specific metrics, the **ctadmn** utility, the c-tree Server's snapshot ability, and the function monitor can be used to identify the specific operations.

   b) Does the system exhibit unexpected system or c-tree Server resource usage patterns (CPU, disk, network usage, etc.) at the time the unexpected performance patterns occur? Use system and c-tree Server utilities to monitor resource usage and compare to normal operation. Differences in resource usage from normal operation may provide insight into the nature of the unexpected performance behavior.

   c) Is the unexpected performance behavior occurring consistently, or does it occur only occasionally? Any pattern that can be identified might help determine the cause of the behavior.

2. Identify any recent changes to the system that might account for the unexpected performance characteristics. For example:

   a) Has the system load changed (for example are more than the usual number of clients using the server or are the clients performing different different operations than usual)?

   b) Have there been any hardware or software changes (including c-tree Server configuration option changes)?

Server Exhibits Unexpected Resource Usage

When c-tree Server or system monitoring tools detect unexpected system or server resource usage, follow these steps to identify the cause of the unexpected resource usage so the problem can be understood and resolved:

1. Identify the nature of the unexpected resource usage as specifically as possible. For example:

   a) Is the resource a system resource or a c-tree Server resource? If a system resource, use system tools to identify the process that is directly responsible for the unexpected resource usage. If the responsible process is the c tree Server, use application and c-tree Server monitoring tools to identify whether activity by particular clients accounts for the change in resource usage. System tools can be used to monitor system calls, dump a core image of the server, or stack traces for server threads. The **ctadmn** utility can be used to terminate clients suspected of contributing to the unexpected resource usage.

   b) Does the unexpected resource usage occur consistently, or does it occur only occasionally? Any pattern that can be identified might help determine the cause of the behavior.

2. Consider whether any recent changes to the system could account for the unexpected resource usage. For example:

   a) Has the system load changed (for example are more than the usual number of clients using the server or are the clients performing different different operations than usual)? Application monitoring tools and the c tree Server's snapshot ability can help identify whether the load on the system has changed recently.

   b) Have there been any hardware or software changes (including c-tree Server configuration option changes)?

Dynamic Dump Fails

A dynamic dump backup may fail for various reasons. The following are some possible causes of a failed dynamic dump:

- The dump script does not exist or is inaccessible.
- The dump script contains invalid options.
- One or more files specified in the dump file list cannot be opened.
- An error occurs when writing the dump stream file (for example, out of disk space or invalid path specified).

The c-tree Server logs dynamic dump error messages and error codes to the server status log, *CTSTATUS.FCS*. In the event of a failed dynamic dump, examine the server status log.

The c-tree Server can be configured to log more detailed dynamic dump progress entries to the server status log, including an entry for each file included in the dump, by adding `DIAGNOSTICS DYNDUMP_LOG` to the server configuration file before starting the server.

Data or Index File Sizes Grow Unexpectedly

The system administrator should monitor the size of c-tree data and index files in order to detect unexpected increases in file size. Monitoring file sizes helps avoid running out of disk space or reaching system file size limits and provides useful information in the event of unexpected server performance behavior or resource usage.

Except for files that are created with deleted space reclamation disabled (using the *ctADD2END* extended create mode), c-tree data and index files reuse deleted space. Fixed-length files that reuse deleted space reuse deleted space with complete efficiency and increase in size only after all deleted space has been reused.

Variable-length c-tree data files reuse deleted space by indexing deleted space by size and storing new variable-length records in the deleted space that most closely matches the size of the new record. The c-tree Server also coalesces adjacent regions of deleted space in variable-length files into a single block of deleted space. Note that a variable-length file may increase in size even if deleted space is available if the available space is not large enough to store a newly-added record. For this reason, depending on the size of variable-length records and the order of insertion and deletion operations on variable-length records, deleted space in variable-length data files can become fragmented over time and the total file size could be larger than might be expected given the total size of active records in the file.

c-tree index files reuse space made available in index nodes by deleted key values and reuse nodes that have become empty and are pruned from the tree. A c-tree index file grows only when a new node is added and no empty nodes remain that can be reused.

Reducing the size of a c-tree data file by removing deleted space from the files can be accomplished by compacting the data file (which also requires rebuilding the associated indexes). The c-tree Plus API function **CompactIFileXtd()** can be used to compact a c-tree data file. Note that a file must be opened in exclusive mode in order to compact it.

To reduce the size of a c-tree index file by removing deleted nodes, rebuild the index file using the c-tree Plus API function **RebuildIFileXtd()**. Note that this function requires exclusive access to the data file and its associated index files. If index file size is a concern, key compress may help reduce the overall index size by storing key values in compressed format. See the *c-tree Plus Programmer's Reference Guide* for details on creating an index that contains compressed keys.

Server Terminates Abnormally

An abnormal server termination is a termination of the c-tree Server process that does not involve a clean server shutdown. The c-tree Server may terminate abnormally for the following reasons:

- The c-tree Server process encounters a fatal exception, causing the system to terminate the server process. In this case the system may produce a core image of the server process at the time of the exception. The c-tree Server status log may contain error messages related to the exception.
- An administrator forcibly terminates the c-tree Server process. In this case, the process is abruptly terminated and the c-tree Server status log does not show an indication of a shutdown occurring.
- The system on which the c-tree Server is running terminates abnormally (due to power loss, operating system exception, or sudden system reboot). In this case, the server process may be abruptly terminated, which case the status log does not show an indication of a shutdown occurring.
- The c-tree Server detects an unexpected internal server error situation known as a catend or a terr. In these cases, the server status log shows an error message containing details about the internal server error.

Recovering From Abnormal Server Termination

It is important to understand the reason for the abnormal server termination so that the appropriate information about the event can be saved and any necessary actions can be taken before restarting the c-tree Server. Follow these steps after an abnormal c tree Server termination occurs:

1. Examine system logs, application logs, and the server status log to determine the nature of the abnormal server termination.
   a) If a fatal exception terminated the server process, save the core file if it exists.
   b) If the server terminated due to a fatal exception or internal c-tree Server error, save a copy of the server's **.FCS* files, the server configuration file, and if time and disk space permit save a copy of all data and index files before restarting the c-tree Server. These files can be used to analyze the abnormal server termination.
   c) If the situation that led to the abnormal server termination can be understood by analyzing the server status log or other system logs, correct the problem that caused the server to terminate. For example, if the server terminated due to insufficient disk space which prevented the server from writing to its transaction logs, free up disk space to ensure the server has enough space for the transaction logs (but do not delete active transaction logs before the server performs its automatic recovery).
2. Determine the status of *PREIMG* and non-transaction data and index files and restore or recover these files as needed. *PREIMG* and non-transaction files are not under full transaction control, so in the event of an abnormal server termination, these files may be in an unknown state. Updates that had been written to the server's cache but not to disk are lost, data files and index files may be out of sync, and *PREIMG* files may be in an inconsistent transaction state.

   To determine if a *PREIMG* or non-transaction file needs to be restored or recovered, open the file using a standalone (non-client/server) c-tree utility. If the file opens successfully, the file is in good shape. If the file open fails with c-tree error 14, the file was updated but was not properly closed and its state is unknown, so the file must be restored or recovered. The options for restoring or recovering *PREIMG* and non-transaction files following an abnormal server termination are the following:
   a) Re-create *PREIMG* and non-transaction files and reload their data from an external source if available, or
   b) Rebuild the files to ensure the data and index files are in sync (although unwritten updates are still lost), or
   c) Restore old copies of the files from backup.

   Note: See the discussion of the *WRITETHRU* filemode and c-tree error 14 for details on the use of *WRITETHRU* and server configuration keywords to avoid error 14 for *PREIMG* and non-transaction files in the event of an abnormal server termination. Be aware that although these options provide

ways to avoid error 14 in such a situation, it is still possible for *WRITETHRU* files to contain data/index inconsistencies or for *PREIMG* files to be in an inconsistent transaction state following an abnormal server termination.

3. Recover *TRNLOG* files using the server's automatic recovery process. After following the above steps, *TRNLOG* files can be recovered by restarting the c-tree Server. The server detects an abnormal server termination and performs automatic recovery of *TRNLOG* files, restoring the *TRNLOG* files to a consistent transaction state. Upon successful completion of automatic recovery, the server is fully operational. At this point clients can connect to the server and can resume their work.

For details on what steps to follow in the event of recovery or restore failures (such as automatic recovery failing), see the section titled "Failures During System Recovery".

4.3 Failures During c-tree Server Shutdown

This section discusses failures that may occur during c-tree Server shutdown.

Server Shuts Down Improperly

This section discusses ways in which a c-tree Server shutdown may fail to complete properly. A normal c-tree Server shutdown is indicated by the following messages in the server status log:

```
Fri Sep 26 14:30:08 2003
 - User# 12     Server shutdown initiated
Fri Sep 26 14:30:09 2003
 - User# 12     Communications terminated
Fri Sep 26 14:30:09 2003
 - User# 12     Perform system checkpoint
Fri Sep 26 14:30:09 2003
 - User# 12     Server shutdown completed
Fri Sep 26 14:30:09 2003
 - User# 12     Maximum memory used was 116088930 bytes
```

A normal c-tree Server shutdown operation may not complete properly for the following reasons:

- The server may terminate abnormally at shutdown due to a fatal exception.
- The server may complete its shutdown without successfully terminating all active client threads.

If the server shutdown terminates abnormally due to a fatal exception, the server does not write the "Server shutdown completed" message to the server status log. Follow the recovery procedures described in the section "Recovering From Abnormal Server Termination".

If the server shutdown completes without successfully terminating all active client threads, the server avoids writing a final checkpoint to the transaction logs. This causes the server to perform automatic recovery on its next startup. The server notes this situation at shutdown by logging the following message to *CTSTATUS.FCS*:

```
Mon Sep 29 16:15:51 2003
- User# 13     Clients active: skipped system checkpoint.
Auto-recovery on next start up
```

Treat this situation similar to an abnormal server termination, because it is possible that clients were modifying *PREIMG* and non-transaction files at the end of server shutdown which could mean that some updates did not get written to disk. Before restarting the c-tree Server, open *PREIMG* and non-transaction files using a standalone utility to determine if they need to be rebuilt. The server's automatic recovery ensures that *TRNLOG* files are in a consistent transaction state on the next server startup.

Server Shutdown Hangs or Takes Excessive Time

The c-tree Server shutdown process may take a long time for the following reasons:

- The server must write all updated data and index cache pages to disk before shutdown completes. If the server is configured to use a large cache, there can be many updated cache pages that must be written to disk at shutdown, which increases the shutdown time.
- The server processes entries in the delete node queue at shutdown. The presence of many entries in the delete node queue can lead to increased server shutdown time.
- The server allows clients time to recognize that the server is shutting down and to disconnect from the server. The shutdown time can be long if many clients are connected or if the server is not able to terminate connected clients.

Monitoring c-tree Server Shutdown Progress

The progress of the c-tree Server shutdown can be monitored in the following ways:

- Monitoring messages the c-tree Server writes to the server status log during shutdown.
- Monitoring messages the c-tree Server writes to its console or standard output during shutdown.
- Monitoring system resource usage by the c-tree Server during shutdown.

The c-tree Server shutdown is normally accompanied by messages in the server status log such as the following:

```
- User# 13     Process delete node Q....
- User# 13     Clients still active.....
- User# 13     Clients shutting down....
```

These messages indicate the current operation the c-tree Server is performing, such as processing delete node queue entries, terminating connected clients, and allowing clients time to shut down.

The server also writes shutdown messages to its console window or standard output. These messages provide more detailed information than the status log entries, including the remaining number of delete node queue entries, and the current number of active client threads:

```
Process delete node Q.......<num_queue> entries.
Clients still active........<num_active>
Clients shutting down.......<num_active>
```

where *<num_queue>* is the current number of entries in the delete node queue and *<num_active>* is the current number of client threads that are still active.

System utilities can also provide insight into the c-tree Server's progress during shutdown. Monitor disk activity on the c-tree data and index files to determine if the server is taking a long time to shut down because it is flushing data and index cache buffers. Monitor the number of active server threads to determine how many client threads are still active. See the "Monitoring c-tree Server Process State" section of this document for additional ways to monitor the state of the c-tree Server process.

Forcibly Terminating the c-tree Server During Shutdown

If the c-tree Server is taking a long time to shut down or if the server appears to hang during shutdown, the server process can be terminated using system utilities but be aware of the effect on c-tree data and index files. Forcibly terminating the c-tree Server process at shutdown effectively causes an abnormal server termination, which means that unwritten updates for *PREIMG* and non-transaction files may be lost and that the server will perform automatic recovery on its next startup in order to ensure *TRNLOG* files are in a consistent transaction state. For details on the state of c-tree data and index files in the event of an abnormal server termination, see the topic "Server Terminates Abnormally" in the "Failures During c-tree Server Operation" section of this document.

4.4 Failures During System Recovery

This section discusses failures that may occur during system recovery.

Automatic Recovery Fails

At startup, the c-tree Server examines the transaction logs to determine whether or not it needs to perform automatic recovery. If so, the server initiates recovery and when the recovery successfully completes, the server startup continues as usual. In some cases, however, the automatic recovery process can fail. For example automatic recovery may fail if:

- The server's transaction logs are damaged, missing, or inaccessible.
- TRNLOG c-tree data or index files that automatic recovery determines it must process are damaged, missing, or inaccessible.
- The server configuration file settings are inconsistent with the settings used the last time the c-tree Server was run.

Recovering from Automatic Recovery Failure

If automatic recovery fails, the c-tree Server logs error messages to its status logs and terminates. In the event of an automatic recovery failure, proceed as follows:

1. Examine the server status log to determine the type of automatic recovery failure.
2. See the specific failure cases in the following sections for details on each type of recovery failure and if possible correct the problem. Restart the c-tree Server and allow automatic recovery to complete successfully.
3. If the automatic recovery failure cannot be corrected, follow these steps to recover or restore *TRNLOG* files and to resume c-tree Server operation:

   a) If automatic recovery terminated due to a fatal exception and the system generated a core file, save a copy of the core file for offline analysis.

   b) Save a copy of the server's transaction logs (**.FCS* files), the server configuration file, and if time and disk space permit save a copy of all *TRNLOG* data and index files. These files can be examined offline to attempt to identify the cause of the automatic recovery failure.

   c) *TRNLOG* files that were in use at the time of the abnormal server termination may be in an unknown state. To determine the state of each *TRNLOG* file, attempt to open each *TRNLOG* file using a c tree file open function. If the file opens successfully, the file is in good shape and did not need to be processed by automatic recovery. If the open fails with error 14, the file must be rebuilt, re-created, or restored from backup.

   Be aware that restoring only some *TRNLOG* files from backup may violate transaction consistency that an application expects to exist among the set of *TRNLOG* files. If this is the case, all related *TRNLOG* files must be treated the same way (all restored from backup for example).

   d) Verify that the server-maintained *TRNLOG* files *FAIRCOM.FCS*, *SYSLOGDT.FCS*, and *SYSLOGIX.FCS* can be properly opened. If the files fail to open (for example, with c-tree error 14 due to the failed automatic recovery) the server will fail to start up.

   FAIRCOM.FCS contains the server's user and group definitions. If *FAIRCOM.FCS* fails to open and the server administrator has not changed the default user and group properties, this file can be deleted and the c-tree Server will re-create it the next time the server starts up. If the server administrator has defined new users and groups or has changed default user or group

properties, this file must be rebuilt or restored from a backup, or the user and group account changes must be re-entered.

SYSLOGDT.FCS and *SYSLOGIX.FCS* contain c-tree Server system event log entries. A copy of these files can be saved and examined offline if desired and the original copy of the files can be removed from the server directory, and the server will re-create these files the next time it starts up.

e) Delete the transaction logs from the server directory. The transaction logs consist of the files *S0000000.FCS, S0000001.FCS*, and all files named *L<lognum>.FCS*, where *<lognum>* is a 7-digit number.

f) Restart the c-tree Server. The server creates a new set of transaction logs and is ready for operation again.

The following sections discuss specific automatic recovery failure situations and the options that are available in each case.

c-tree File Open Errors During Recovery

Automatic recovery fails if a *TRNLOG* file that must be processed during recovery cannot be opened. See the section titled “Errors Occur When Opening c-tree Files” for possible errors that may occur when opening c-tree files and what can be done in each case.

A special case that can occur during automatic recovery is that a file cannot be opened because an application used a c-tree Plus API function to delete or rename the file. In some cases, automatic recovery does not realize that the file should be expected to be missing for this reason, and automatic recovery attempts to open the file and fails with c-tree error 12 because a file by the specified name does not exist.

Note: Creating *TRNLOG* files as transaction-dependent files (in which file creation and deletion are guaranteed to be atomic and these events are indicated by transaction log entries) avoids most occurrences of this type of situation, but does not guarantee that this situation cannot occur.

When this situation occurs, the server logs the following messages to the server status log:

```
Tue Sep 30 10:44:10 2003
 - User# 01     mark.idx: 12
Tue Sep 30 10:44:10 2003
 - User# 01
     *** Recovery may proceed by adding 'SKIP_MISSING_FILES YES' ***
     *** to the server configuration file.    ***
Tue Sep 30 10:44:10 2003
 - User# 01    Automatic recovery terminated with error: 12
```

As indicated in the server status log messages, the `SKIP_MISSING_FILES` server configuration keyword can be added to the server configuration file in order to avoid this error when a file is missing during automatic recovery. Because error 12 can occur for other reasons (for example, a file may be inaccessible to the c-tree Server process due to file permissions or due to an unavailable volume), confirm that the specified file does not exist and that there is a reasonable explanation as to why this file does not exist before adding the `SKIP_MISSING_FILES` option to the server configuration file and restarting the c-tree Server.

Automatic Recovery Terminates Abnormally

If the c-tree Server process encounters a fatal exception during automatic recovery, causing the system to terminate the server process, the system may produce a core image of the server process at the time of the exception. The c-tree Server status log may contain error messages related to the exception.

In this situation, examine the server status log to see if there are any error messages that point to the cause of the exception. If the status log shows automatic recovery errors, consult the appropriate

section above for actions based on the specific error code shown in the status log. The c-tree Server may be restarted in order to retry automatic recovery, but if the recovery continues to fail in this manner and the problem cannot be corrected, follow the steps listed in the "Recovering From Automatic Recovery Failure" section.

Automatic Recovery Takes Excessive Time

Automatic recovery may take a long time to complete for the following reasons:

- Server configuration settings such as increasing the log size and checkpoint interval may require the server to scan a significant amount of log entries and to process a considerable number of transaction undo and redo operations.
- *TRNLOG* indexes that do not use the *LOGIDX* filemode may need to have their tree structure reconstructed, which can increase recovery time.

In the event of a long automatic recovery, server administrator has the following options:

- Allow automatic recovery to complete (waiting as long as it takes).
- Terminating the server process and restarting recovery (if server configuration settings or other system properties can be changed that may improve recovery speed).
- Terminating the server process and abandoning recovery (re-creating or restoring *TRNLOG* files from backup).

See the "Server Startup Hangs or Takes Excessive Time" section for details on monitoring automatic recovery progress and the “Recovering from Automatic Recovery Failure” topic above for steps to follow when choosing to abandon automatic recovery and re-create or restore *TRNLOG* files.

Dynamic Dump Restore Fails

Restoring files from a dynamic dump stream file may fail for various reasons. Some examples include:

- The dump restore script is missing.
- The dump restore script contains invalid options or options that are inconsistent with the c-tree Server settings.
- An error occurs when the dump restore attempts to restore files to the dump time.
- The dump restore is performed in a directory with files that interfere with the restore procedure (such as existing transaction logs).

When a dump restore operation fails, the **ctrdmp** utility logs error messages to the file *CTSTATUS.FCS*. Check this file for a c-tree error code that explains the cause of the failure and take the appropriate action. Review the dump restore procedure to ensure that the proper steps were followed.

Data and index files are dumped to the dump stream file by reading the contents of the file from disk. Because a file is not instantaneously read in its entirety, a data or index file in the dump stream file may consist of the file contents as they appear over a period of time. Dump recovery includes the transaction log activity during the dump time, so that the file can be restored to its state at the time the dump began. For this reason, if the dump restore successfully extracts files from the dynamic dump stream file but fails when attempting to restore the files to the dump time, the restored data and index files are in an unknown state.

If **ctrdmp** fails when attempting to restore the files to the dump time and no solution can be found, the affected data and index files can be rebuilt to ensure the data and index files are in sync, but the rebuild may fail because the data file may contain a mixture of record images from different points in time, or the files can be restored from a different backup.

Note: Consider performing dump restore operations offline immediately after the dynamic dump backup is performed so that dump restore failures can be resolved at backup time rather than at restore time.

File Rebuild Fails

A file rebuild operation may fail for various reasons. Examples include:

- The *IFIL* resource in the data file is missing or damaged.
- The data file cannot be opened (for example, if the header of the file is corrupted).
- The rebuild detects illegal duplicate key values.

When a rebuild fails, check the return code of the c-tree API function used to rebuild the file and consult the *c-tree Plus Function Reference Guide* entry for that function to determine the appropriate action.

The **ctrbldif** utility can be used to rebuild a file that contains a valid *IFIL* resource. The utility opens the data file and retrieves the *IFIL* resource from the file. If the utility cannot read the *IFIL* resource, it prompts for the name of a file containing a good copy of the *IFIL* resource. Consider saving a copy of the file containing the *IFIL* resource for use in such a situation.

The **ctrbldif** utility and c-tree Plus rebuild API functions support an option to handle the presence of illegal duplicate keys by marking records containing duplicate key values as deleted.

If a file rebuild fails and no solution is found which allows the rebuild to complete successfully, re-create the file and reload the data from an external source if available, or restore a backup copy of the file.

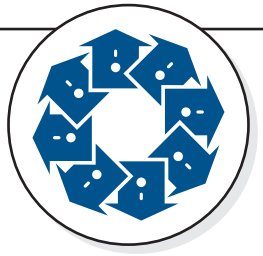
File Compact Fails

Like a file rebuild operation, a file compact operation may fail for various reasons. When a file compact operation fails, check the return code of the c-tree API function used to compact the file and consult the *c-tree Plus Function Reference Guide* entry for that function to determine the appropriate action.

The **ctcmpcif** utility can be used to compact a file that contains a valid *IFIL* resource. The utility opens the data file and retrieves the *IFIL* resource from the file. If the utility cannot read the *IFIL* resource, it prompts for the name of a file containing a good copy of the *IFIL* resource. Consider saving a copy of the file containing the *IFIL* resource for use in such a situation.

The **ctcmpcif** utility and c-tree Plus compact API functions support an option to handle the presence of illegal duplicate keys by marking records containing duplicate key values as deleted.

If a file compact fails and no solution is found which allows the compact to complete successfully, re-create the file and reload the data from an external source if available, or restore a backup copy of the file.



Appendix A. ctstat Utility Reference

A.1. ctstat - Statistics Utility

The c-treeACE Server Statistics Utility, **ctstat**, is a client utility used to display statistics collected by the c-treeACE Server. **ctstat**, provides valuable real time monitoring of critical c-treeACE Server operations.

Usage

```
# ctstat  report_type [-s svn] [-u uid] [-p upw]
        [-i int [cnt]] [-h frq] [-d] [-m] [-t]
```

reports:

-vas	Admin-System Report
-vts	Tivoli-System Report
-vaf file...	Admin-File Report
-vtf file...	Tivoli-File Report
-vau user...	Admin-User Report by User Name
-vau handle...	Admin-User Report by User Handle
-vah handle...	Admin-User Report by Conneciton Handle
-func	Function Timing Report
-funcfile	Function Timing By File Report
-userinfo	User Report with stats from *USERINFO()* function
-isam	ISAM Activity Report
-sql	SQL Activity Report
-text	System Activity Report, Write System Snapshot to *SNAPSHOT.FCS.*
-file [csv]	File Activity Report
-iotime on | off	Turn disk I/O call timing on or off
-wrktime on | off	Turn function call timing on or off

options:

-s svn	c-treeACE 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. The following additional statistics are output: • *phyrec* - Last byte offset of file for non-memory file or current memory in use for memory file. • *mhghbyt* - Largest amount of memory used for memory file since file was created. • *memcnt* - Current number of memory records. • *hghcnt* - Largest number of memory records since file was created.
-t	Output timestamp with header.

Admin-System Report Example

The admin-system (*-vas*) report displays c-treeACE Server system-wide statistics in the areas of cache usage, disk I/O, open files, established client connections, file locks, and transactions.

Example

Below is a sample admin-system report produced by executing the command:

```
ctstat -vas -u ADMIN -p ADMIN -s FAIRCOMS -h 10 -i 2
```

```
     cache     disk i/o   files   connect        locks         transactions
d%h %m i%h %m  r/s  w/s  cur/max  cur/max   cur l%h %m  dead act t/s  r/t  w/t
 99  1  99  1    1    1 2581/10000   2/128     0 100  0     0   1  36    0    0
 99  1  99  1    2    0 2581/10000   2/128     0 100  0     0   1  35    0    0
 99  1  99  1    2    0 2581/10000   2/128     0 100  0     0   1  28    0    0
 99  1  99  1    0    0 2581/10000   2/128     0 100  0     0   1  22    0    0
```

The columns shown in this report are described as follows:

```
d%h   Data cache hit rate
%m    Data cache miss rate [100 - Data cache hit rate]
i%h   Index cache hit rate
%m    Index cache miss rate [100 - Index cache hit rate]
r/s   Disk reads per second
w/s   Disk writes per second
cur   Current number of open files
max   Server limit on number of open files
cur   Current number of client connections
max   Server limit on number of client connections
cur   Number of locks currently held
l%h   Lock hit rate [(lock attempts - locks blocked - locks denied) / lock
      attempts]
%m    Lock miss rate [100 - Lock hit rate]
dead  Number of dead locks detected
act   Current number of active transactions
t/s   Number of transactions per second
r/t   Number of read operations per transaction
w/t   Number of write operations per transaction
```

Tivoli-System Report Example

The Tivoli-system (*-vts*) report displays c-treeACE Server system-wide statistics in the areas of cache usage, disk I/O, open files, established client connections, file locks, and transactions. The Tivoli-system report displays much of the same statistics that the admin-system (*-vas*) report displays, but in a format appropriate for input to tools such as the Tivoli monitoring application.

Example

Below is a sample Tivoli-system report produced by executing the command:

```
ctstat -vts -u ADMIN -p ADMIN -s FAIRCOMS -h 10 -i 2

#%cachehit %cachemiss r/s w/s maxfiles openfiles totalconnections activetransactions numdeadlock
trans-r/s trans-w/s %hashhit %hashmiss transactions/s
92 8 0 0 10000 18 1 0 0 0 0 100 0 0
92 8 0 9 10000 18 1 0 0 0 17 100 0 1
92 8 0 0 10000 18 1 0 0 0 0 100 0 1
92 8 0 0 10000 18 1 0 0 0 0 100 0 1
92 8 0 1 10000 18 1 0 0 0 1 100 0 1
92 8 0 0 10000 18 1 0 0 0 0 100 0 1
```

Note: The header line shown in this example is written as a single output line although it may be shown on multiple lines here.

The columns shown in this report are described as follows:

```
%cachehit           Data and index cache hit rate
%cachemiss          Data and index cache miss rate
r/s                 Disk reads per second
w/s                 Disk writes per second
maxfiles            Server limit on number of open files
openfiles           Current number of open files
totalconnections    Current number of client connections
activetransactions  Current number of active transactions
numdeadlock         Number of dead locks detected
trans-r/s           Number of read operations per transaction
trans-w/s           Number of write operations per transaction
%hashhit            Lock hit rate
%hashmiss           Lock miss rate
transactions/s      Number of transactions per second
```

Admin-File Report Example

The admin-file (*-vaf*) report displays c-treeACE Server statistics for the specified file. Note that multiple data or index files can be specified on the command line. Below is a sample admin-file report produced by executing the command:

```
ctstat -vaf mark.dat mark.idx -u ADMIN -p ADMIN -s FAIRCOMS -h 10

r/s  w/s entries locks l%h %m dlock recrd  node t filename
   2    4   11863     4 100  0     0   128   n/a F mark.dat
   0    4   11863     2  96  4     0   n/a 32768 I mark.idx
   1    3   12192     4 100  0     0   128   n/a F mark.dat
   0    9   12192     3  97  3     0   n/a 32768 I mark.idx
   2    4   12730     5 100  0     0   128   n/a F mark.dat
   0    3   12730     1  97  3     0   n/a 32768 I mark.idx
   2    4   13236     5 100  0     0   128   n/a F mark.dat
   0    2   13236     0  97  3     0   n/a 32768 I mark.idx
```

The columns shown in this report are described as follows:

```
r/s       Disk reads per second for the file
w/s       Disk writes per second for the file
entries   Number of data records or key values in file
locks     Number of locks currently held on file
l%h       Lock hit rate for the file
%m        Lock miss rate for the file
dlock     Number of dead locks detected for the file
```

```
recrd     Record length if data file, otherwise n/a
node      Node size if index, otherwise n/a
t         File type (F=fixed-length data, V=variable-length data, I=index)
filename  Name of the file
```

Tivoli-File Report Example

The Tivoli-file (*-vtf*) report displays c-treeACE Server statistics for the specified file in a format appropriate for input to tools such as the Tivoli monitoring application.

Below is a sample Tivoli-file report produced by executing the command:

```
ctstat -vtf mark.dat mark.idx -u ADMIN -p ADMIN -s FAIRCOMS -h 10
```

```
#r/s w/s currentlocks waitinglocks filename
0 0 5 0 mark.dat
0 0 1 1034 mark.idx
1 3 4 0 mark.dat
0 6 0 1120 mark.idx
3 5 5 0 mark.dat
0 0 0 1208 mark.idx
2 4 4 0 mark.dat
0 0 2 1324 mark.idx

2 4 5 0 mark.dat
0 3 2 1402 mark.idx
```

The columns shown in this report are described as follows:

```
r/s             Disk reads per second for the file
w/s             Disk writes per second for the file
currentlocks    Number of locks currently held on file
waitinglocks    Cumulative lock wait count
filename        Name of the file
```

Admin-User Report Example

The admin-user report, *-vau user...*, displays c-treeACE Server statistics for the specified users. All existing connections whose user ID match the specified user ID are displayed.

Example

Below is a sample admin-user report produced by executing the command:

```
ctstat -vau GUEST -u ADMIN -p ADMIN -s FAIRCOMS -h 10
```

```
log function   sec fil lok l%h %m dlock tid/uid/nodename
 7s TRANEND      0   2   1  98  2     0 GUEST/10/
 7s ADDREC       0   2   2  98  2     0 GUEST/12/
 7s ADDREC       0   2   1  98  2     0 GUEST/13/
 7s ADDREC       0   2   0  98  2     0 GUEST/14/
 0s ctSNAPSHOT   0   2   0   0  0     0 ADMIN/15/ctstat
 7s ADDREC       0   2   0  98  2     0 GUEST/17/
```

The columns shown in this report are described as follows:

```
log                Total logon time for client
function           Function client is currently executing
sec                Current function request time
fil                Current number of files open by this client
lok                Current number of locks held by this client
l%h                Lock hit rate for this client
%m                 Lock miss rate for this client
dlock              Number of deadlocks detected for this client
tid/uid/nodename   Server thread ID/User ID/Node name for this client
```

Function Timing Report Example

The function timing report (*-func*) displays c-treeACE Server statistics for each c-tree function that a client has called at least once since the time the server started.

Below is a sample function timing report produced by executing the command:

```
ctstat -func -u ADMIN -p ADMIN -s FAIRCOMS -h 10

function              counter       secs
FRSVSET                    10      0.002
NXTVSET                    20      0.001
GETDODAX                    2      0.000
COMMBUF                     1      0.000
ctSNAPSHOT                 10      0.002
```

The columns shown in this report are described as follows:

```
function  Function name
counter   Cumulative number of times this function has been called
secs      Cumulative elapsed time for this function
```

Text Report Example

```
ctstat -text -u ADMIN -p ADMIN -s FAIRCOMS -h 10
```

This command writes a c-treeACE Server system snapshot to the file *SNAPSHOT.FCS*. See the file *SNAPSHOT.FCS* for the detailed server statistics.

I/O Time Statisics Example

The c-treeACE Server *SNAPSHOT* feature includes support for collecting disk read and write timings on a per-file basis when high-resolution timer support is activated. Use the **ctstat** utility's *-iotime* option to toggle the collection of disk I/O timings.

- Turn on disk I/O timings:
  ```
  # ctstat -iotime on -u ADMIN -p ADMIN -s FAIRCOMS
  ```
- Turn off disk I/O timings:
  ```
  # ctstat -iotime off -u ADMIN -p ADMIN -s FAIRCOMS
  ```

The **ctstat** utility's *-vaf* option also outputs differential I/O timings for each file when the c-treeACE Server returns version 2 (or higher) *GFMS* structure statistics.

Example

```
C:\> ctstat -vaf mark.dat mark.idx -h 1 -i 10
 r/s  w/s  read time  write time  entries locks l%h %m dlock recrd  node t filename
   0    0          0           0    26239     1 100  0     0   128   n/a F mark.dat
   0    0          0           0    26239     0  99  1     0   n/a  8192 I mark.idx
 r/s  w/s  read time  write time  entries locks l%h %m dlock recrd  node t filename
 128  237          1          12   108309     2 100  0     0   128   n/a F mark.dat
   0    2          0           0   108308     0  99  1     0   n/a  8192 I mark.idx
 r/s  w/s  read time  write time  entries locks l%h %m dlock recrd  node t filename
 121  243          1          14   186164     2 100  0     0   128   n/a F mark.dat
   2   27          0           4   186163     0  99  1     0   n/a  8192 I mark.idx
 r/s  w/s  read time  write time  entries locks l%h %m dlock recrd  node t filename
 109  219          1          10   256356     2 100  0     0   128   n/a F mark.dat
3247 3296         39          77   256355     0  99  1     0   n/a  8192 I mark.idx
 r/s  w/s  read time  write time  entries locks l%h %m dlock recrd  node t filename
 103  206          1          10   322381     4 100  0     0   128   n/a F mark.dat
5623 5640         67         124   322380     1  99  1     0   n/a  8192 I mark.idx
```

I/O Statisics per File Example

The c-treeACE Server *SNAPSHOT* feature supports a mode to write snapshot statistics for all files open by the c-treeACE Server to disk. Use the **ctstat** utility's *-file* option. The snapshot statistics for all open files are then written to the *SNAPFILE.FCS* file in comma-delimited or human-readable format.

CSV Example

Write statistics for all open files to *SNAPFILE.FCS* in comma-delimited format using the **ctstat** utility:

```
# ctstat -file csv -i 1 1
```

Sample SNAPFILE.FCS Contents

```
On-Demand File Snapshot
Mon Jun 25 16:40:51 2007

physical file size,logical file size,serial number,active entries,tran high mark,update
timestamp,max file size,read ops,bytes read,write ops,bytes written,memory file high bytes,read
time (msec.),write time (msec.),index height,file id,server id,time id,node size,record
length,permanent file mode,max leaf key bytes,max non-leaf key bytes,file type,key length,key
member number,number of members, super file type,max leaf marks,wrthdr sequence number,total lock
attempts,header lock attempts,total lock wait count,header lock wait count,deadlocks,total locks
denied,total locks freed,total blocks released,current locks held,current blocked requests,max
special cache pages,current special cache pages,number of buffer pages,number of data cache
pages,number of channels,number of users with file open,current memory record count,highest memory
record count,killed locks,max segments,active segments,update flag,file type,duplicate key
flag,index delete type,key padding byte,flavor,alignment,pointer size,file name
16384,16384,0,15,0,0,0,0,0,5,16768,0,0,0,-1,0x00000000,0x00000000,0x00000000,8192,0,0x0000000
0,8148,8174,1,12,0,0,0,2048,3,44,23,0,0,0,0,44,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0xff,0,0,0,32,2,8,
4,I0000001.FCS

====================
```

Human Readable Example

Write statistics for all open files to *SNAPFILE.FCS* in human-readable format using the **ctstat** utility:

```
# ctstat -file -i 1 1
```

Sample SNAPFILE.FCS Contents

```
On-Demand File Snapshot
Tue Jun 26 09:58:28 2007

      phyrec     numrec     sernum    nument    hghtrn    tstamp    mxfilz   fredops     fredbyt
fwrtopts      fwrtbyt      mhghbyt    fredtim    fwrttim idxhgt     fileid     servid     timeid
nodsiz   reclen     logtyp maxkbl maxkbn filtyp keylen kmem nmem suptyp maxmrk   hdrseq  floktry
flokhlk  flokblk  flokhbk  flokdlk  flokdny  flokfre  flokrel  flokcur  fblkcur   datlmt   datspl
bufcnt   datcnt   numchn  fusrcnt   memcnt   hghcnt  flokkil   segmax   seglst updflg  ktype autodup
deltyp keypad flflvr flalgn flpntr filename
       16384       16384           0        15         0         0         0         1        8192
10       57728           0          0          0     -1 0x00000000 0x00000000 0x00000000     8192
0 0x00000000   8148   8174      1     12    0    0      0   2048        3       68       35        0
0        0        0       68        0        0        0        0        0        0        0        1
1        0        0        0        1        0   0xff      0       0      0     32       2      8
4 I0000001.FCS

====================
```

Existing Connections Userinfo Example

A report option, *-userinfo*, is available to display additional statistics about existing user connections.

New information included in this alternative output:

- The status and idle time of the connection.

- The last c-treeACE Server request made.
- Time spent in a transaction.
- Amount of memory consumed by the client.
- Number of files open by the client.
- The time the user has been logged in.
- User ID, Thread ID, and Nodename of the user.

Example

```
# ctstat -vau -u ADMIN -a ADMIN FAIRCOMS


    status  lastrequest trntime   mem fils   time uid/tid/nodename
    -- idle --          --        10K    0     -- dumpit.script/13/DYNAMIC DUMP
   59s idle OPNRFIL     --        37K    9    59s ADMIN/16/
    -- idle --          --        10K    0     -- dumpit.script/13/DYNAMIC DUMP
01m00s idle OPNRFIL     --        37K    9 01m00s ADMIN/16/
    -- idle --          --        10K    0     -- dumpit.script/13/DYNAMIC DUMP
01m01s idle OPNRFIL     --        37K    9 01m01s ADMIN/16/
    -- idle --          --        10K    0     -- dumpit.script/13/DYNAMIC DUMP
01m02s idle OPNRFIL     --        37K    9 01m02s ADMIN/16/
    -- idle --          --        10K    0     -- dumpit.script/13/DYNAMIC DUMP
01m03s idle OPNRFIL     --        37K    9 01m03s ADMIN/16/
    -- idle --          --        10K    0     -- dumpit.script/13/DYNAMIC DUMP
01m04s idle OPNRFIL     --        37K    9 01m04s ADMIN/16/
```

ISAM Statistics Example

The c-treeACE Server system *SNAPSHOT* includes counters for ISAM record add, delete, update and read operations. The **ctstat** utility includes an *-isam* option which displays various ISAM counters.

Example

```
# cstat -isam -u ADMIN -p ADMIN -s FAIRCOMS

     add  delete  update    read    total
   10216   10215       0   10215    30646
   10113   10114       0   10114    30341
   10147   10146       0   10146    30439
   10164   10165       0   10165    30494
   10070   10069       0   10070    30209
```

Enable Function Call Times by File

The c-treeACE Server *SNAPSHOT* support collects c-tree function call counts and timings on a per-c-tree file basis. This support enhances the c-treeACE Server's existing support for collecting c-tree function call counts and timings, which are collected as totals for all files. Enabling collection of c-tree function timings now enables collection of both the total and file-specific function timings.

The **ctstat** utility includes a *-wrkstat* option to enable the collection of this data.

Example

Turn on function call timings:

```
# ctstat -wrktime on -u ADMIN -p ADMIN -s FAIRCOMS
```

Turn off function call timings:

```
# ctstat -wrktime off -u ADMIN -p ADMIN -s FAIRCOMS
```

Function call Times by File Example

The c-treeACE Server *SNAPSHOT* function supports a mode that writes function timings for all files open by the c-treeACE Server to disk. Use the **ctstat** utility's *-funcfile* option to output these timing statistics for all open files to *SNAPFUNC.FCS* in either a comma-delimited format, or a human readable format:

CSV Example

```
# ctstat -funcfile csv -i 1 1
```

SNAPFUNC.FCS Contents

```
On-Demand Function Snapshot
Tue Jun 26 13:22:40 2007

DELVREC,DELVREC,ADDREC,ADDREC,RWTREC,RWTREC,GETALTSEQ,GETALTSEQ,SETDEFBLKX,SETDEFBLKX,GTEVREC
,GTEVREC,GETSEG,GETSEG,GETMAP,GETMAP,GTEREC,GTEREC,ADDVREC,ADDVREC,GETDODAX,GETDODAX
count,time,count,time,count,time,count,time,count,time,count,time,count,time,count,time,count
,time,count,time,count,time,filename
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,I0000001.FCS
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,D0000000.FCS
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,FAIRCOM.FCS
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,FAIRCOM.FCS!USER.dat
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,FAIRCOM.FCS!USER.idx
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,FAIRCOM.FCS!GROUP.dat
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,FAIRCOM.FCS!GROUP.idx
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,FAIRCOM.FCS!UG.dat
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,FAIRCOM.FCS!UG.idx
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,FAIRCOM.FCS!UG.idx M#01
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,FAIRCOM.FCS!UVAL.dat
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,FAIRCOM.FCS!UVAL.idx
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,SYSLOGIX.FCS
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,SYSLOGDT.FCS
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,SYSLOGIX.FCS M#01
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,SYSLOGIX.FCS M#02
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,D0000001.FCS
0,0,590737,502096,698884,347088,0,0,2,0,0,0,0,0,5,0,0,0,0,0,20,0,mark.dat
0,0,0,0,0,0,5,0,0,0,0,0,0,5,0,0,0,0,322988,136944,0,0,0,0,mark.idx
0,0,0,0,0,0,5,0,0,0,0,0,0,5,0,0,0,0,376483,171571,0,0,0,0,mark.idx M#01
0,0,0,0,0,0,5,0,0,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0,mark.idx M#02

====================
```

Human-readable Example

```
# ctstat -funcfile -i 1 1
```

SNAPFUNC.FCS Contents

```
On-Demand Function Snapshot
Wed Jun 27 15:26:06 2007

                ADDREC                RWTREC                GTEREC
     count         time      count         time      count         time  filename
         0            0          0            0          0            0  I0000001.FCS
         0            0          0            0          0            0  D0000000.FCS
         0            0          0            0          0            0  FAIRCOM.FCS
         0            0          0            0          0            0
         0            0          0            0          0            0  FAIRCOM.FCS!USER.dat
         0            0          0            0          0            0  FAIRCOM.FCS!USER.idx
         0            0          0            0          0            0  FAIRCOM.FCS!GROUP.dat
         0            0          0            0          0            0  FAIRCOM.FCS!GROUP.idx
         0            0          0            0          0            0  FAIRCOM.FCS!UG.dat
         0            0          0            0          0            0  FAIRCOM.FCS!UG.idx
         0            0          0            0          0            0  FAIRCOM.FCS!UG.idx M#01
         0            0          0            0          0            0  FAIRCOM.FCS!UVAL.dat
         0            0          0            0          0            0  FAIRCOM.FCS!UVAL.idx
         0            0          0            0          0            0  SYSLOGIX.FCS
```

```
           0           0           0           0           0           0  SYSLOGDT.FCS
           0           0           0           0           0           0  SYSLOGIX.FCS M#01
           0           0           0           0           0           0  SYSLOGIX.FCS M#02
           0           0           0           0           0           0  D0000001.FCS
        9910       74592        1380       10993           0           0  mark.dat
           0           0           0           0        1381       67772  mark.idx
           0           0           0           0           0           0  mark.idx M#01
           0           0           0           0           0           0  mark.idx M#02

====================
```

Function Timing Limitations

- The function timings for a file are reset to zero when the file is physically closed.
- As currently implemented, the c-tree function timings do not track c-tree API function calls made by c-treeACE SQL clients.

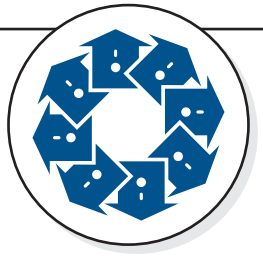
Memory File Usage Example

The **ctstat** utility supports an option, *-m*, that when specified with the *-vaf* report option, causes **ctstat** to output the following additional memory file statistics:

- *phyrec -* Last byte offset of file for non-memory file or current memory in use for memory file.
- *mhghbyt* - Largest amount of memory used for memory file since file was created.
- *memcnt* - Current number of memory records.
- *hghcnt* - Largest number of memory records since file was created.

Example

```
# ctstat -vaf disk.dat mem.dat -h 1 -i 2 -m
 r/s  w/s entries locks l%h %m dlock recrd  node t     phyrec    mhghbyt   memcnt   hghcnt filename
   0    0     n/a     0 100  0     0    15   n/a V 1923110761 1923110761    19232    19232 mem.dat
   0    0       3     0   0  0     0   128   n/a F       4096          0        0        0 disk.dat
```



Appendix B. ctsysm Utility Reference

This appendix describes how to use the **ctsysm** utility to monitor c-tree Server status log entries.

B.1. c-tree Server Status Log Monitoring Utility

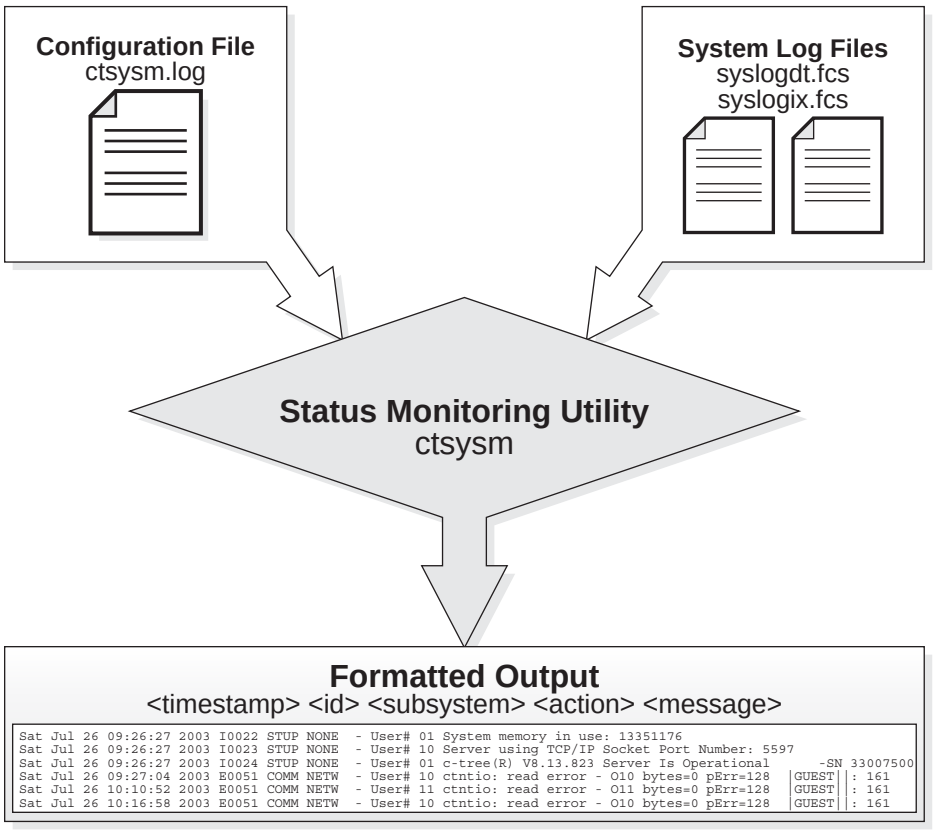
The **ctsysm** utility can be used to monitor the system log for status log messages. To use this utility the FairCom server must log messages to the system log. This can be achieved by adding the following keyword to the server's configuration file:

```
SYSLOG CTSTATUS
```

With this keyword in place, the server logs each entry in the *ctstatus.fcs* file to the system log files (*syslogix.fcs* and *syslogdt.fcs*). The utility is independent of the version of the server so it is not necessary update the server in order to use **ctsysm**.

The utility can read the system log from the beginning or it can save its position and start again from that set position by specifying the -f command line option.

Figure 17: System Monitorying Utility



B.2. ctsysm Configuration File

The configuration file (default name: *ctsysm.log*) classifies the messages by specifying a list of subsystems that originated the log entry, the recommended actions, and details for each message. The configuration file is simply a text file.

The subsystem list consists of entries in this format:

```
<subsystem_keyword> <subsystem_description>
```

The action list consists of entries in this format:

```
<action_keyword> <action_description>
```

The messages are in the following format:

```
<code> <subsystem> <action> <num_lines> <text>
```

where <code> is a 5-character code beginning with a message type and followed by a 4-digit message number (for example, E0001). Possible message codes include:

- F: Fatal error
- E: Error
- W: Warning
- I: Information
- U: Unclassified

Because the configuration file is a text file separate from the server executable, it can easily be updated as new information about specific messages becomes available, without having to update the server executable.

B.3. ctsysm Output

The **ctsysm** utility outputs messages in the following format:

```
<timestamp> <code> <subsystem> <text>
```

For example:

```
Sat Jul 26 09:26:27 2003 Innnn STUP NONE  - User# 01 System memory in use: 13351176
```

B.4. ctsysm Command-Line Usage

The following is the appropriate command line usage of the c-tree Server Status Monitoring Utility:

```
ctsysm [-s svn] [-u uid] [-p upw] [-r rpt] [-c cfg] [-f fil] [-e]

 -s svn   c-tree Server name
 -u uid   user name
 -p upw   user password
 -r rpt   repeat interval
 -c cfg   config file
 -f fil   save/restore state to file
 -l log   status log name (SYSLOGDT.FCS or CTSTATUS.FCS)
 -e       direct ctsysm error messages to standard output in the same format as messages read
from the server status log.
```

-s, -u, and -p specify the server name, user name, and user password used when the utility connects to the c-tree Server.

-c specifies the name of the utility's configuration file (default is *ctsysm.cfg*).

-f causes the utility to saves the current state of the utility to the specified file when the utility shuts down and to read the current state of the utility from the specified file when starting up. This option can be used to cause the utility to start reading after the last message it read from the system log instead of starting at the beginning of the system log.

-l specifies the name of the file to read, either *SYSLOGDT.FCS* (for monitoring server status using the server's system log) or *CTSTATUS.FCS* (for monitoring server status using *CTSTATUS.FCS* directly). Using *SYSLOGDT.FCS* has the advantage that it supports the -f option and can be used from a remote system, while using *CTSTATUS.FCS* has the advantage that all status log messages are captured (note that the server may not log some startup and shutdown messages to the system log because the system log is not active during part of startup and shutdown; perhaps the most critical time of interest that the system log is not active is during automatic recovery).

When *-e* is used, an error message output by **ctsysm** is of the form:

```
<timestamp> E0000 SYSM MSGT ctsysm ctsysm <error message text>
```

Index

R

S

T

U

V

W