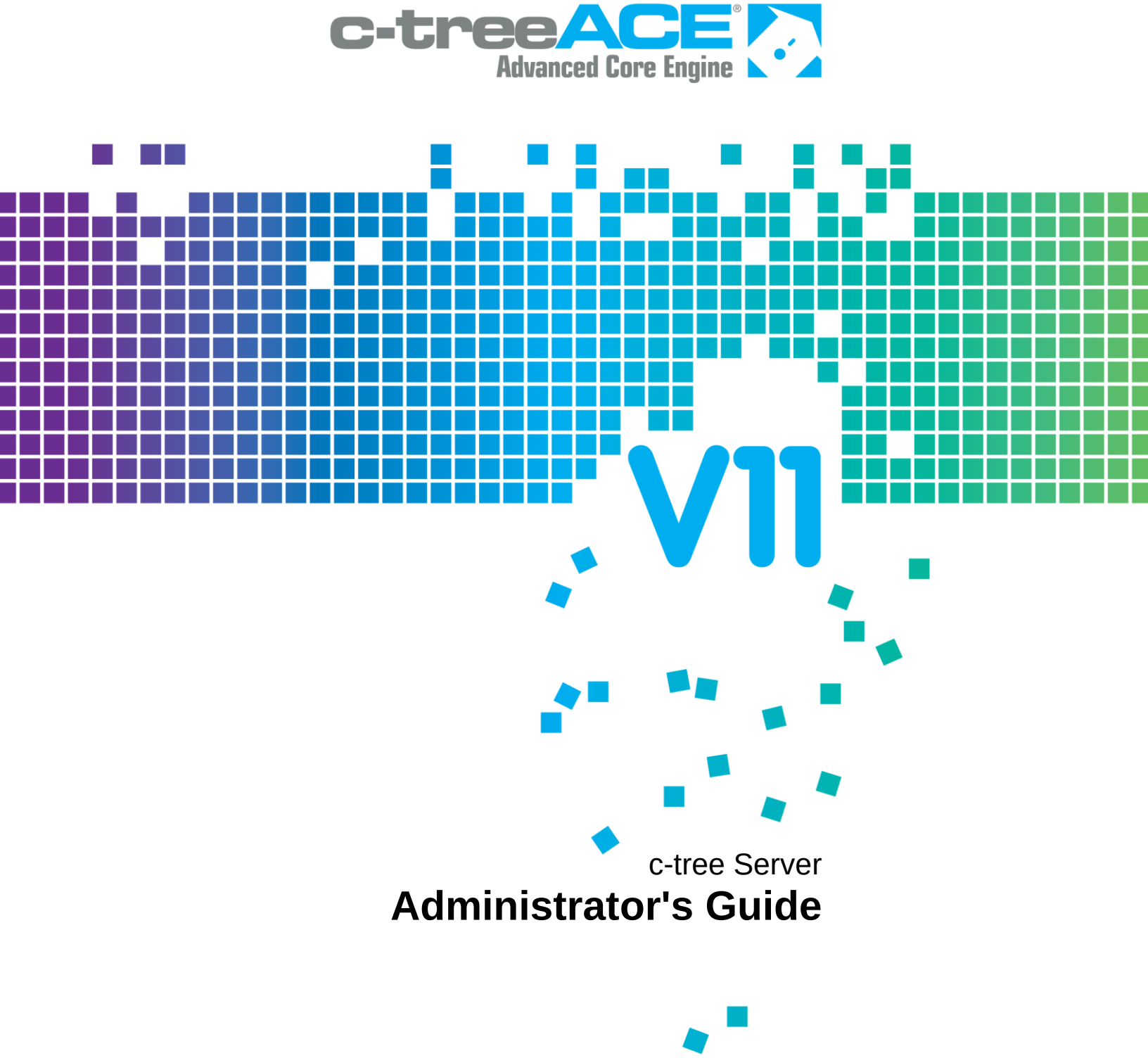


c-tree Server

Administrator's Guide

c-tree Server

Administrator's Guide

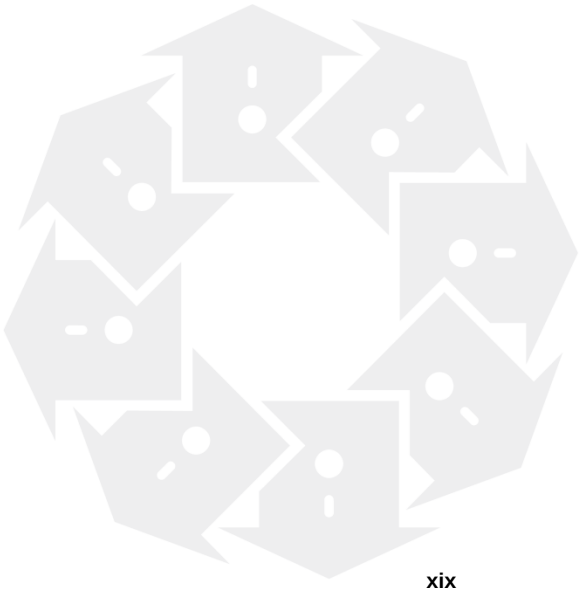
Copyright Notice

12/6/2016

Contents

Table of Figures

No table of figures entries found.



FairCom Typographical Conventions

Before you begin using this guide, be sure to review the relevant terms and typographical conventions used in the documentation.

The following formatted items identify special information.

Formatting convention	Type of Information
Bold	Used to emphasize a point or for variable expressions such as parameters
CAPITALS	Names of keys on the keyboard. For example, SHIFT, CTRL, or ALT+F4
FairCom Terminology	FairCom technology term
FunctionName()	c-treeACE Function name
Parameter	c-treeACE Function Parameter
`Code Example`	Code example or Command line usage
utility	c-treeACE executable or utility
filename	c-treeACE file or path name
`CONFIGURATION KEYWORD`	c-treeACE Configuration Keyword
CTREE_ERR	c-treeACE Error Code

1. c-treeACE Server Administrator's Guide

1.1 Server Quick Start

The c-treeACE Server is designed for ease of use with minimal administration. This is different from other database server products, which require extensive setup, administration, and support.

To make the c-treeACE Server operational requires virtually no effort:

1. Install
2. Execute and Run!

Install

Follow the installation instructions provided with your c-treeACE distribution media.

Execute

Execute the **ctsrvr** executable (*ctsrvr.exe* on Windows). This runs the c-treeACE Server with the default settings.

That's all it takes to get your server up and running.

While the c-treeACE Server runs properly right out-of-the-box, the rest of this guide details the installation, operation, and optional configuration settings available to the Server Administrator. FairCom recommends using the available security options, establishing regular data backup procedures, and optimizing the server configuration for your environment to maximize performance.

See "Introduction" (page 1) for an overview of the content of this guide.

1.2 Introduction

This manual has two main purposes for the c-treeACE Server Administrator:

1. To provide a quick, easy way to see what responsibilities you have, and
2. To provide the information needed to manage c-treeACE Server operation.

The c-treeACE Server supports high-level database management, including:

- **client/server computing** - Increases performance and provides the ability to maintain database integrity, especially in multi-user environments. The basic principle of client/server computing is: applications, or "clients", interact with the server, which manages file operations and communicates with clients.

- **online transaction processing (OLTP)** - The c-treeACE Server can group a specified set of operations, called a “transaction,” and ensure either all of them are done or, if there is a problem, none will be done, e.g., either all of an invoice is processed, or none of it.
- **security controls** - c-treeACE Server access is controlled with user IDs, passwords, file permissions, and encryption. Users and files may be added to Administrator defined “groups”, e.g., shipping department, payroll department.
- **database maintenance and utilities** - The c-treeACE Server automatically saves necessary information for use in automatic or Administrator-specified backups and recovery from problems.
- **configuration flexibility** - From basics such as which communication protocol the c-treeACE Server uses and memory allocations to enforce for specific users, to a wide range of advanced controls.

Further technical details concerning the c-treeACE Server are available in "Overview of the c-treeACE Server" and other FairCom documents.

The c-treeACE Server Administrator has the following six areas of responsibility, each of which could be divided among several people:

Installation

Someone, not necessarily the Administrator, must physically load the c-treeACE Server software onto the computing environment. Once completed, installation issues usually are no longer a concern unless the c-treeACE Server needs to be re-installed, for example, to install a new version. See "c-treeACE Server Installation" (page 7) for details.

Operating the Server

Starting and stopping the c-treeACE Server: Any user can start the c-treeACE Server by running the executable module, **ctsrvr**, as any other program in the environment. See "Operating the c-treeACE Server" (page 35) for details.

Controlling access to the c-treeACE Server

Begin by setting up valid User IDs and passwords (including your own). Establish rules of access to given database files. Establish groups where users and files can be associated and control access according to membership in those groups.

Use **ctadmn**, the c-treeACE Server Administration Utility, to control access with user IDs, file passwords, file permissions, and Administrator defined groups with specified access rights to particular files. This utility also monitors user status and/or disconnects users from the c-treeACE Server.

ctpass is used by the Administrator or any other authorized user to change the password associated with their User ID.

ctfile is used by the Administrator or any user to change file security information on any file owned by the user. See "Controlling c-treeACE Server Access" (page 44) for details.

Maintaining Database Integrity

Schedule and conduct backups or dumps of system generated files for later use in recovering from problems or returning a database to its status at a prior time.

Use the utility **ctdump** to schedule dynamic dumps that can be used at a later time to restore database files or to roll back to a state at a previous point in time.

ctrdmp works with information saved in a dynamic dump to either recover from a catastrophic system failure by restoring specified files to a consistent, well-defined state or to roll back specified files to their state at a specified time.

Use the utility **ctfdmp** to recover from a catastrophic failure using a previously saved dynamic dump or complete backup, which may be made using any standard backup utility. This allows you to restore backups then 'roll forward' to a given time using preserved log files.

(For programmers) Use the utility **ctldmp** to carry out a transaction log dump, which records partial log related information, for use in application development. See "Maintaining Database Integrity" (page 92) for details.

Configuring the c-treeACE Server

Understand how the c-treeACE Server is currently configured and, optionally, change configuration settings (e.g., to set memory allocation limits, to select communication protocols, to activate a particular dump description script).

The c-treeACE Server is started by any user authorized to start **ctsrvr**. Routine starting of the c-treeACE Server is not necessarily a major responsibility for the Administrator.

The User ID "ADMIN" (default password is "ADMIN") and members of the ADMIN group are the only users who can access **ctstop**, the utility for stopping the c-treeACE Server, so stopping the c-treeACE Server is always a major Administrator responsibility.

Customize the c-treeACE Server

No configuration file is required, but if the c-treeACE Server is to be reconfigured to replace any default settings, a file named *ctsrvr.cfg* must be created for the server to load at startup. See "Configuring the c-treeACE Server" (page 153) for details.

Note: Utility names and methods of executing them may vary slightly in different environments, so see the individual sections in this manual for specifics. The utilities covered here are not the only ways to carry out Administrator duties and the utilities listed here are not necessarily the only ones available.

The basic topics covered here are for orientation only. "c-treeACE Server Installation" (page 7) and "Operating the c-treeACE Server" (page 35), are considered required reading for c-treeACE Server Administrators. "c-treeACE Server Access Configuration" (page 44), "Maintaining Database Integrity", "Configuration File Format" (page 153), and "Basic Keywords" (page 161) are recommended reading. The rest of "c-treeACE Configuration Options" (page 160) is optional and intended for advanced users.

Some issues may require the assistance of others with specialized knowledge relevant to the operating environment (e.g., configuring memory access allotments, defining dynamic dumps).

Additional information can be found in the following appendices:

- "User's Control of Security Options" (page 54) contains user password control information.
- "Glossary" (page 362) defines terms.

1.3 Advanced c-treeACE Server Features

By choosing c-treeACE database technology, you obtain high performance, absolute data integrity, and scores of advanced features solving many database challenges. Here are just some of the advanced options available:

- ACID Transaction Processing Engine for Complete OLTP Support
- Automatic Transaction Recovery
- Robust Multithreaded Performance
- Secure Client Connections
- Replication
- Dynamic "Hot' Backups
- Roll-forward restorations
- Advanced Data Encryption
- Data Compression
- Memory Files
- Partitioned Files
- Deadlock Free Locking
- Advanced Data and Index Cache Controls
- Small Footprint
- Administrative Tools

1.4 Client/Server Computing

Client/server computing removes most of the difficult and tedious issues of database management from application programs and assigns these operations to a separate program, called a data server, which operates between the application program and its data.

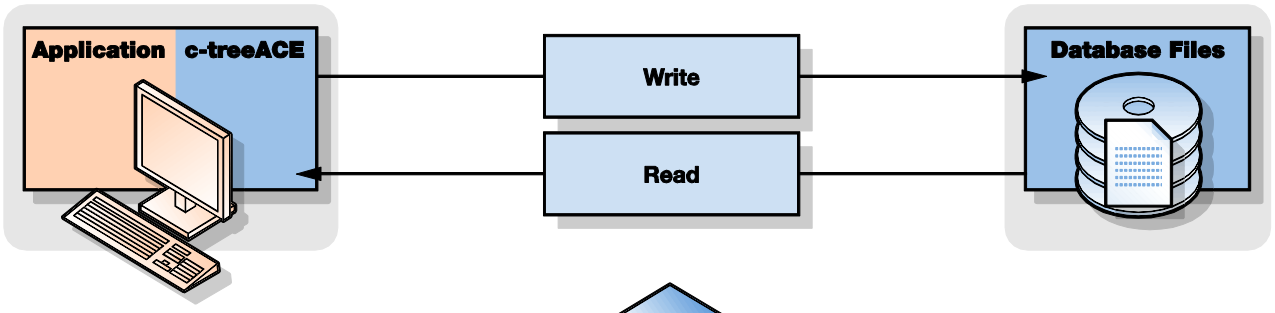
In client/server computing, application programs are clients making requests of a server, which goes to the relevant files, executes all operations needed to carry out the request, and sends back a response to the client application.

To implement this design, an application program needs to communicate with the database server. Exact details about which information-processing tasks are carried out by the application and which are carried out by the server depend on the server involved and on the “client-side” data management code used in the application.

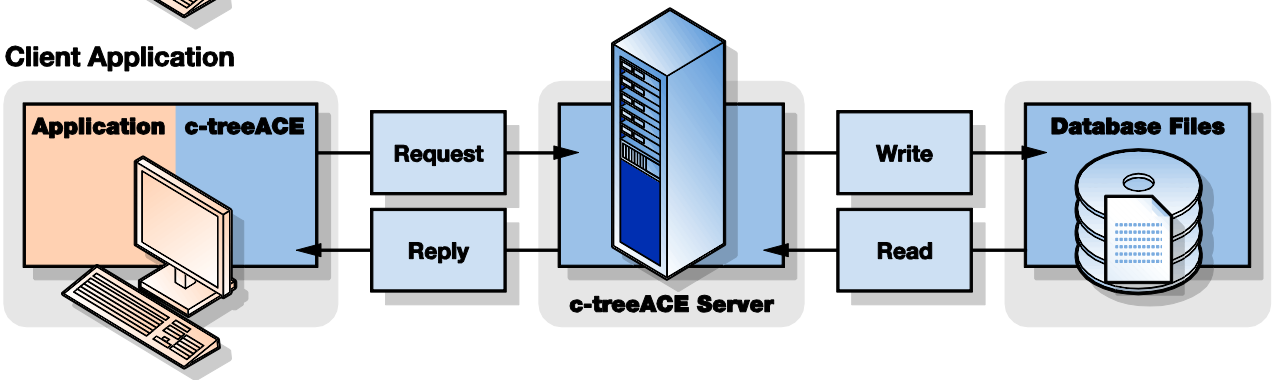
The following diagrams illustrate how client/server computing is different from direct database manipulation.

In the standalone method of database operations, applications deal with files directly, using functions supplied by a third party (e.g., the c-treeACE multi-user non-server library), or user supplied functions. All responsibility for security, coherence, and speed of access, in a single user or a multi-user environment, are the responsibility of the application code using data management functions.

Standalone Application



Client Application



In the client/server architecture, database operations function in a client application program interfaced to a database server. The database server contains the “intelligence” needed to process requests from clients, interact with the relevant database files, wherever they are located, and respond to those requests.

The database server in client/server computing plays a number of roles all of which add power to applications with a minimum of effort, including:

- Minimizes the flow of information from one place to another. By processing and responding to communication “request” and “response” messages between application programs and database files, the data server eliminates the need to send whole files of information from place to place. Only relevant data moves across the network.
- Manages multi-user issues. The server manages requests so users don’t get in one another’s way, or create inconsistencies in the database.
- Coordinates sending and receiving of information over networks even where database files and/or applications reside on different types of machines and operating systems (heterogeneous support).
- Implements transaction processing (see "Transaction Processing" in the "c-treeACE Programmers Reference Guide" (http://docs.faircom.com/doc/ctreeplus/)).
- Implements security features (described in "c-treeACE Server Access Configuration" (page 44)).
- Offers absolute data integrity through file mirroring.

Client/server computing is more and more important for the most basic of reasons: it offers increased speed, control, and efficiency in data management.

2. c-treeACE Server Installation

Installing the c-treeACE Server is mostly a matter of copying software from the distribution media onto the system. c-treeACE installation can be completed in three simple steps:

1. Install the c-treeACE Server and support utilities.
2. License c-treeACE with an appropriate License Authorization File.

   Note: Your application vendor may provide licensed c-treeACE Server.

3. Start c-treeACE.

The following sections provide the information necessary for installing the c-treeACE Server in specific environments. Before proceeding, verify that the computer on which c-treeACE is to be installed has sufficient capacity for the c-treeACE Server and associated applications.

The “Minimum Hardware Requirements” sections discuss the minimum memory (RAM) requirements of the c-treeACE Server not including operating system memory requirements. Additional RAM for file caching, opening files, supporting many users, etc., is encouraged for optimal performance and functionality. "Server Memory Calculations" (page 41) provides formulas for approximating c-treeACE Server memory requirements.

The hard drive space specifications contained in the following sections indicate the minimum space necessary to install the c-treeACE Server on each particular operating system or platform.

Skip to the appropriate operating system section where your c-treeACE Server is to be installed.

2.1 License Authorization File

License Object

Beginning with c-treeACE V10.0 an activation process is no longer required to use the c-treeACE Server. A much simpler approach requiring only the presence of a new **License Authorization File** is now used. There are substantial advantages to this new approach. Primarily, it avoids an extra process to use the server and handling lengthy key values. More importantly, it avoids modifying any binary files, which can break package checksums, for example.

The **License Authorization File** is a binary file containing unique licensing information assigned by FairCom. This licensing information permits the c-treeACE Server technology to operate on a specified operating system, to support specific features, to support a fixed number of concurrent users and/or connections to the c-treeACE Server technology, and to utilize a fixed number of CPUs on the host machine.

The license file is named *ctsrvr-<SN>.lic* where *<SN>* is the unique Serial Number assigned to your server instance and provided by FairCom. This file will need to be properly placed in the same directory where the c-treeACE Server binary is located, for example, *\FairCom\V11.x.x\<platform>\bin\ace\sql\ctreesql.exe*. An example developer license is shown below:

```
<?xml version="1.0" encoding="us-ascii"?>
<ctlicense version="2">
  <version>11</version>
  <serial>39000010</serial>
  <OEM>1</OEM>
  <lictype>Development</lictype>
  <cpus>2</cpus>
  <servtype>ALL (Standard - SQL)</servtype>
  <users>32</users>
  <private>EDE . . . QGP</private>
  <checksum>LE47LG9DNM06IA30CGAFFO00NMCEJL59</checksum>
</ctlicense>
```

Note that you can read the most relevant sections of this XML file in plain text regarding serial numbers, connection and CPU counts. If this licensing file isn't present, you'll receive a 960 error in your *CTSTATUS.FCS* status log upon startup:

```
"LICENSE ERROR: License initialization failed: Missing license file."
```

The Developer edition of the c-treeACE Server included with the c-treeACE Professional package includes a *ctsrvr-<SN>.lic* file configured to support up to 32 concurrent connections and can operate on up to 2 concurrent CPU cores (as reported by the operating system which may include physical CPUs, CPU cores, or virtual CPUs assigned to a partition).

> Development servers are licensed exclusively for development and testing purposes and only by the developer who is the c-treeACE Professional license holder. They are expressly not authorized for production use. Should you need additional licenses for testing or if you wish to test with a license file supporting a greater number of connections or CPUs, please contact your nearest FairCom office.

When purchasing a production c-treeACE Server license, you will receive a *ctsrvr-<SN>.lic* file via e-mail, along with a "Proof of Entitlement" document that summarizes the configuration of your c-treeACE Server license file.

Activation of Servers Prior to V10.0

Servers prior to c-treeACE V10.0 required a one-time activation process which required an activation key and serial number.

Execute **fcactvat** and follow the prompts to activate the server executable file. The c-treeACE Server activation process stamps the server executable for the number of concurrent connections specified by the server license purchased.

2.2 c-treeACE Server for Windows

Operational Environment

The c-treeACE Server for Windows uses the Windows multi-threading functions and is named *ctreesql.exe* (or *ctsrvr.exe*). The c-treeACE Server for Windows is distributed with support for the following communication protocols:

Protocol	COMM_PROTOCOL Keyword
TCP/IP	F_TCPIP
TCP/IP (using an IPv6 socket)	F_TCPIPV6
Shared Memory	FSHAREMM
TCP/IP (Data Camouflage support)	FETCPIP

The c-treeACE Server for Windows defaults to the TCP/IP protocol. To activate any other protocol, use the `COMM_PROTOCOL` keyword in a *ctsrvr.cfg* file. Use the name shown in the table above as the token following the `COMM_PROTOCOL` keyword in *ctsrvr.cfg*.

Windows Note: c-treeACE Server V10.3 and later automatically detect if the client is on the same machine as the server. If they are on the same machine, the shared memory protocol will automatically be used, provided the `COMM_PROTOCOL FSHAREMM` protocol is active.

Note: The `COMM_PROTOCOL` option specifies the protocol used for ISAM connections. By default, local SQL connections use shared memory unless the `SQL_OPTION NO_SHARED_MEMORY` keyword is specified. See the `COMM_PROTOCOL` (`http://docs.faircom.com/doc/ctserver/#27910.htm`) for more information about the communication protocol for SQL connections.

The `COMM_PROTOCOL` keyword disables the default protocol, so if you want to load the default and another protocol, each must have a `COMM_PROTOCOL` entry in *ctsrvr.cfg*. For example, to load all supported communication protocols for the c-treeACE Server on Windows Server 2003, add the following lines to *ctsrvr.cfg*:

```
COMM_PROTOCOL   F_TCPIP
COMM_PROTOCOL   FSHAREMM
COMM_PROTOCOL   FETCPIP
```

Note: If `COMM_PROTOCOL` is specified for one protocol, all protocols to be used must be specified. If no `COMM_PROTOCOL` is specified, the c-treeACE Server uses the default, F_TCPIP.

The shared memory protocol eliminates the overhead of the TCP/IP protocol stack resulting in very fast communications. The only drawback is the client and server must reside in the same physical memory space. For client server applications running on the same machine, this can result in communications performance increases of almost 500% over TCP/IP in some instances.

Communication DLLs Removed from the c-treeACE Windows Server

Please note that, as part of a recent security enhancement effort, beginning with c-treeACE V11.2 and c-treeRTG V2, packages for Windows no longer contain the following DLLs:

- F_TCPIP.DLL
- F_TCPIPV6.DLL
- FETCPIP.DLL
- FSHAREMM.DLL

These communication modules are now statically linked into the server binary as of c-treeACE V11.2 and c-treeRTG V2. Non-Windows servers have always been statically linked. As a result, these Windows DLLs are no longer required in the c-treeACE Server directories.

Installing as a Windows Service

Microsoft Windows supports background processes known as services that are handled somewhat differently by the operating system. Services may be configured to start automatically at system startup or to start manually by a user. Services have no user interface and can continue to run even when no users are logged on to the system. The operating system automatically terminates services at system shutdown or a user can manually terminate them.

FairCom's c-treeACE Server is compatible with Windows Service support.

The c-treeACE Server *.msi* installers will configure the c-treeACE Server process to operate under the Windows Service Manager by default. To manually install c-treeACE as a Windows service, use the Windows **sc.exe** command.

Example

```
C:> sc  create  "c-treeACESQL"  binPath= "C:\install_path\ctreesql.exe"  start= auto
DisplayName= "c-treeACE SQL Database Engine"
```

Tip! A space is required after binPath=, start=, and DisplayName=

The c-treeACE service features all of the capabilities and advantages of Windows services described above. As with any service, the c-treeACE service can be configured to start automatically when the machine comes up, can run invisibly no matter which users are logged on or if no user is logged on, and will shut down automatically when the host machine shuts down.

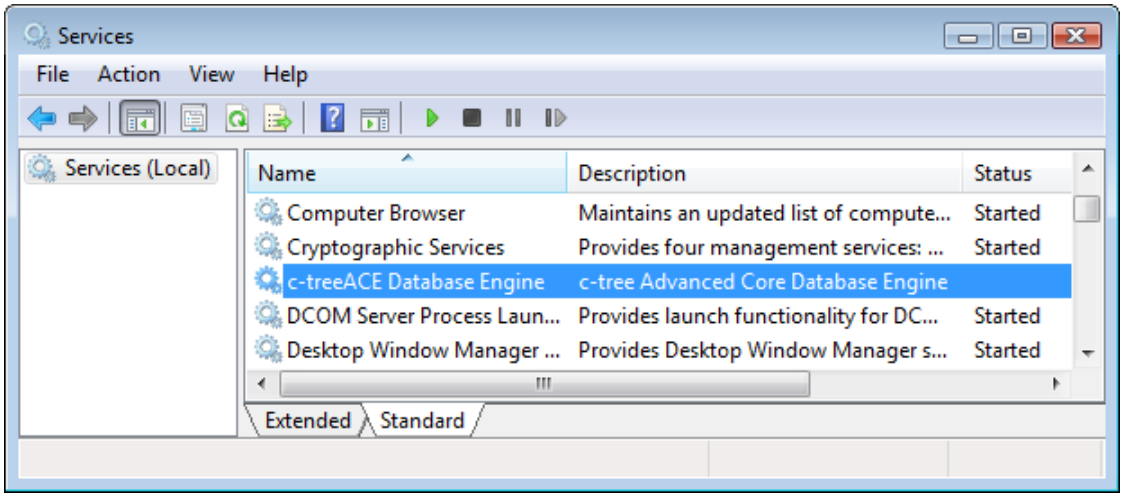
Configuring the c-treeACE Service

The c-treeACE service has two configurable properties: the Startup Type and Logon User.

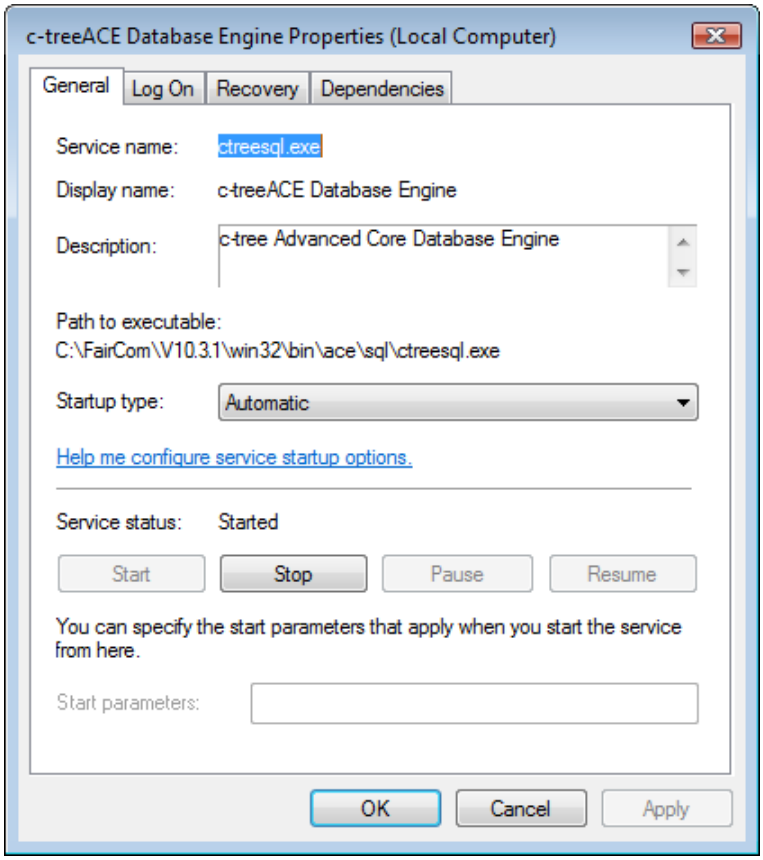
The Logon User and the Startup Type (see the following Figure; Windows Service configuration options) can be set using the Windows Services Control Panel applet. To do so:

1. Open the Windows Control Panel by clicking **Start > Settings > Control Panel**.

2. Select the Services applet. You will be presented with a list of the installed services (see the following figure, the Services applet in the Windows Control Panel):



3. Select the c-treeACE service, then double-click it or right-click and choose **Properties**.
4. The service configuration options window will appear (see the following figure, the Windows Service configuration option). Set the Startup Type and Logon User, as desired.



Note: The Microsoft Windows Services Control Panel applet is the preferred method for service configuration, and other administration, such as starting/stopping the service. The c-treeACE **ctntinst.exe** command no longer ships with the product as of V10.3.

Starting the c-treeACE Service

Start the c-treeACE service using either the Microsoft **sc.exe** command or the Windows Services Control Panel applet.

To start the c-treeACE service using the Windows Services Control Panel applet:

1. Open the Control Panel.
2. Select the Services applet.
3. From the list of the installed services (see Figure "Windows Services Control Panel applet"), select the name for the c-treeACE service (“c-treeACE Database Engine” by default).
4. Click **Start** to start the c-treeACE service.

Displaying the current status of the c-treeACE Service

The current status of the c-treeACE service can be determined by using either the Microsoft **sc.exe** command or the Windows Services Control Panel applet.

To check the current status of the c-treeACE service using c-treeACE Server SCP, run the following (where "`ctreesql.exe`" is the name of the c-treeACE service):

```
sc query ctreesql.exe
```

```
C:\FairCom>sc query ctreesql.exe

SERVICE_NAME: ctreesql.exe
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                              (STOPPABLE, NOT_PAUSABLE, ACCEPTS_PRESHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

To check the current status of the c-treeACE service using the Windows Services Control Panel applet: Open the Control Panel and select the Services applet. If the c-treeACE service is running, the Status field shows “Started”. Otherwise the Status field is blank.

Stopping the c-treeACE Service

The c-treeACE service can be stopped by using either the Microsoft **sc.exe** command or the Windows Services Control Panel applet, or the net stop command.

Example

```
C:>  net stop "c-treeACE SQL"
```

- The service stops automatically when Windows signals the operating system itself is shutting down. A clean shutdown of Windows should result in a clean shutdown of the c-treeACE service. However, since Windows only allows a 20-second delay for service shutdown, FairCom recommends all files be maintained under transaction processing to allow automatic recovery if cache cannot be safely flushed to prevent data corruption. The default 20-second delay can be adjusted using the **WaitToKillServiceTimeout** registry key, found in **HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control**, when present.
- Windows Vista and later allow a new Service PRESHUTDOWN notification, which gives the service extra time to complete its shutdown. This can help to avoid a lengthy auto-recovery if the server has too many cache pages to flush within the SHUTDOWN limit.

Removing the c-treeACE Service

If you wish to remove the c-treeACE service from the list of installed Windows Services, the Microsoft general service controller utility, **sc.exe**, can be used. See the Microsoft website for information about using this command-line utility.

Service Troubleshooting Tips

This section identifies possible problems that may be encountered when using c-treeACE as a Windows service, and ways to diagnose and solve them.

Problems starting the c-treeACE Service

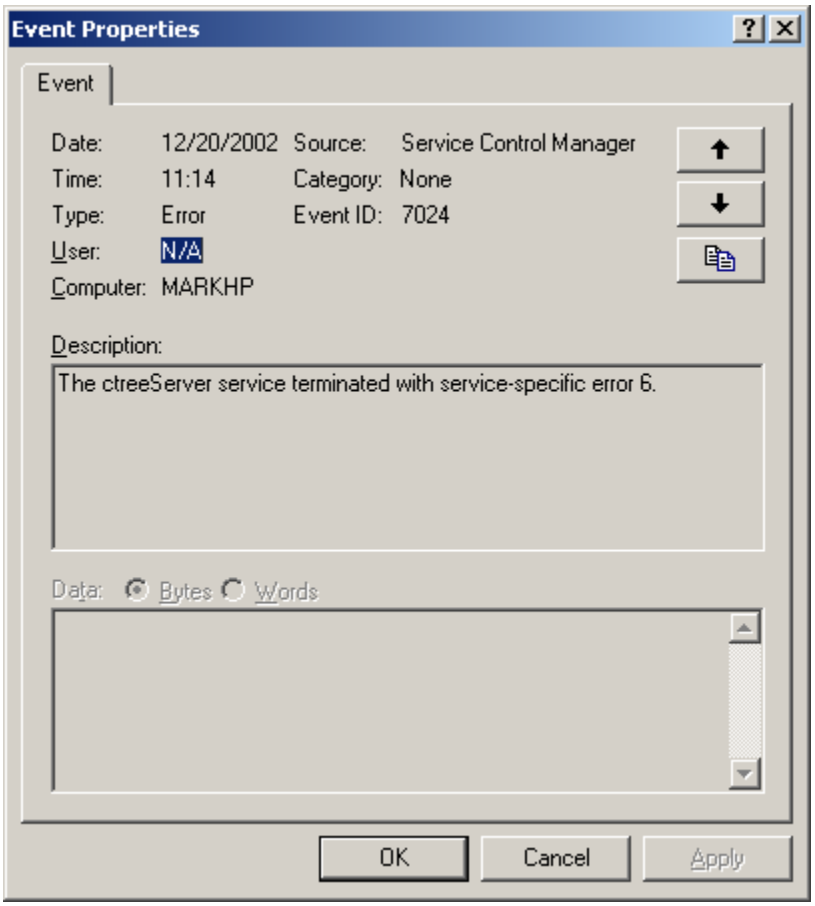
If the c-treeACE service fails to start, it returns a service-specific error, and logs a message to the Windows application event log. This information can be used to determine the reason the c-treeACE Server service failed to start. Below is the output of a failed startup when starting the c-treeACE service using FairCom's SCP. The service-specific error is displayed as the "Service Exit" code.

```
Starting the c-tree Server service...
c-treeServer start unsuccessful:
Current State: STOPPED
Win32 Exit:    1066
Service Exit:  6
Checkpoint:    0x0
WaitHint:      0x0
```

The table below shows possible service-specific errors returned by the c-treeACE Server service, the corresponding message, and possible causes for each of these errors.

Error Code	Error Message	Possible Causes
2	(Varies)	An operating system function call failed. See the event log for a detailed error message.
4	The c-treeACE settings file is missing. It is required to operate this server.	c-treeACE service requires a settings file, but it was not found.
5	The current settings file is invalid.	c-treeACE requires a settings file, and the settings file that was found was not valid. Contact your application developer for assistance.
6	c-treeACE must be activated with a FairCom activation key to operate. See the c-treeACE Activation Key Card within your package for more information.	c-treeACE has not yet been activated.

Use the Windows Event Viewer to list events reported by the c-treeACE service. Start the Event Viewer and select the Application log option from the Log menu. Events logged by the c-treeACE service have the “Source” field set to the service name (“c-treeACE Database Engine” by default). Double-clicking an event displays the event detail (see the following Figure; Using the Event Viewer to display events logged by the c-treeACE service).



Problems connecting to the c-treeACE Service

If client applications are unable to connect to the c-treeACE service, verify that c-treeACE service is running (See "Displaying the current status of the c-treeACE Service" (page 12) for details).

If the c-treeACE service is running, check the c-treeACE status log file (*CTSTATUS.FCS*, typically located in the directory in which the c-treeACE executable resides) for the following information:

1. Are there any error messages logged to *CTSTATUS.FCS*?
2. Is the Server Name displayed in *CTSTATUS.FCS* the same Server Name your client applications are using?
3. Are the protocols displayed in *CTSTATUS.FCS* the same as those your client applications are using?

FairCom’s **ctadmn** utility (provided with the c-treeACE Server) is an additional useful tool for verifying whether clients can connect to c-treeACE.

Problems stopping the c-treeACE Service

If you are unable to stop the c-treeACE service, check the event log for an error message. Also check for error messages in the c-treeACE status log file (*CTSTATUS.FCS*, typically located in the directory in which c-treeACE executable resides).

FairCom's **ctadmn** utility (provided with the c-treeACE Server) can also be used to stop the c-treeACE service.

Minimum Hardware Requirements for c-treeACE V10

c-treeACE SQL Server (SQL Version)

The minimum CPU and memory requirements for operating the SQL version of the c-treeACE SQL Server for Windows are:

- Pentium 1 GHz CPU
- 512 MB RAM
- 500 MB Disk space + space for your data + index files (assuming default 120MB transaction LOG_SPACE setting in *ctsrvr.cfg*)

c-treeACE Server (ISAM Version)

The minimum CPU and memory requirements for operating the ISAM version of the c-treeACE Server for Windows are:

- Pentium 300 MHz CPU
- 300 MB RAM
- 250 MB Disk space + space for your data + index files (assuming default 120MB transaction LOG_SPACE setting in *ctsrvr.cfg*)

Note: Additional memory will be needed for additional users beyond 16 concurrent users and larger data and index caches. 1+ GB of RAM is recommended for c-treeACE SQL Server.

Cache sizes larger than 2GB require a 64-bit version of the OS and c-treeACE.

Minimum Software Requirements for c-treeACE

c-treeACE V10.4 and later require the following:

- Windows XP/2003 or newer
- Microsoft .NET Framework 4 or newer.
- For the c-treeACE SQL JDBC Driver: To develop using JDBC, you will need JDK 1.6 or newer and the c-treeACE SQL Server.
- Stored procedures for the c-treeACE SQL Server:
  - To develop a Java stored procedure, you will need JDK 1.6 or newer.
  - To execute a Java stored procedure, you will need JRE 1.6 or newer.

Java Version

c-treeACE SQL V10 and V11 require the Java 1.6 JDK and JRE environment for Stored Procedures, Triggers, and User Defined Scalar Functions (UDFs), JDBC, and c-treeDB Java. Java is readily available from the Oracle Java downloads (http://www.oracle.com/technetwork/java/javase/downloads/index.html) website. c-treeACE supports Java on any platform that the Java environment is currently available, including Windows, AIX, Oracle Sun, and Linux.

Note that Oracle has announced an end of life (http://www.oracle.com/technetwork/java/javase/eol-135779.html) policy for Java 1.6 beginning February 2013. Check the FairCom http://www.faircom.com/ web site for the latest Java compatibility announcements and availability of the latest Java support.

.NET Framework Requirements

c-treeACE V10 requires at least Framework Version 3.5 SP1 complete (e.g., the complete version, not just the "client" version).

c-treeACE V10.4 and later requires Microsoft .NET 4.0 Framework.

ADO.NET Entity Framework V2 - V4 Support

The c-treeACE SQL ADO.NET Data Provider has support for Entity Framework V2 through V4 (for EF6, see *Entity Framework 6 Support* in the *c-treeACE SQL ADO.NET Data Provider http://docs.faircom.com/doc/ado_net/* manual). When Visual Studio 2008 or later is detected during c-treeACE Professional installation, support is integrated by default.

System Requirements

The minimum development system requirements for c-treeACE SQL ADO.NET Entity Framework support are:

1. Visual Studio 2008 Service Pack 1
2. Microsoft .NET Framework 3.5 Service Pack 1

Auto Incrementing Field Type Restriction

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

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

Other Known Limitations

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

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

c-treeACE Server for Windows Installation

1. Insert the c-treeACE Server CD into the proper drive.
2. The installation Setup utility starts automatically. Follow the instructions to install the c-treeACE Server. If Setup does not start automatically, execute it from the CD.

During installation, communication DLL files will be placed in the c-treeACE Server directory. These c-tree communication DLLs can be accessed by leaving them in the same directory as the c-treeACE Server executable or by placing them in a directory referenced by the PATH environment variable.

For c-treeACE versions prior to V10.0, the installation setup utility should automatically attempt to activate the c-treeACE Server using the **fcactvat** program. See your c-treeACE Server Activation Key Card for instructions. Some c-treeACE Server OEM vendors may provide pre-activated c-treeACE Server with their applications.

Tool Tray Interface

When the server configuration file contains the `CONSOLE TOOL_TRAY` keyword, the c-treeACE Server starts in background, displaying only a c-treeACE icon in the Windows tool tray. This feature is especially nice for ‘simple’ user sites, with no system administrative expert. Although more sophisticated sites will prefer running the c-treeACE Server as a service, this feature gives a similar ‘service-like’ background effect, without the user needing to learn Windows service administration.

Add the following keyword to your server configuration file, *ctsrvr.cfg*:

```
CONSOLE   TOOL_TRAY
```

This keyword is not supported when the server is running as a service.

The c-treeACE Server for Windows accepts the ‘&’ symbol, (“^&” for Windows Server 2003/XP/Vista), as a command line parameter to execute in `CONSOLE TOOL_TRAY` mode. The following example launches the server in “background-tool-tray” mode:

```
C:\server>  ctreesql &
```

or

```
C:\server>  ctsrvr &
```

2.3 c-treeACE Server for Mac

Operational Environment

This version of the c-treeACE Server is designed specifically to work on the Apple Mac platform. Applications using this release communicate via the TCP/IP protocol. Mac client processes can execute on the same machine as the c-treeACE Server for Mac.

The c-treeACE Server for Mac is shipped on the c-treeACE Server CD, which contains the c-treeACE Server executable, **ctreesql** or **ctsrvr**, and the utility and companion programs discussed throughout this guide.

Minimum Hardware Requirements

The minimum CPU required by the c-treeACE Server for Mac is a G3 processor or later. The RAM required to operate the c-treeACE Server for Macintosh is the RAM required by the operating system plus 4MB RAM for up to 8 users; 8MB for more than 8 users. Requires Mac OS X 10.2 or later. (Intel, Motorola and Universal Binary builds available.)

The minimum hard drive space required by the c-treeACE Server for Mac is:

The size of the c-treeACE Server executable
+ the amount specified by the `LOG_SPACE` keyword (10 MB default)
+ 1MB for c-treeACE Server status logs
+ 2MB for the pre-compiled c-treeACE Server utilities
+ the size of the data (*.dat*) and index (*.idx*) files

This operating system supports files larger than 4 GB.

Configuring Mac Systems

Mac systems may have special configuration requirements. In addition to configuring c-treeACE, the operating system itself may need to be configured. For example, the number of file descriptors must be large enough to accommodate the number of files c-treeACE will access, which could be larger than the operating system's default setting.

User Limits (ulimit)

- File descriptors should be set to a number greater than:

  `FILES` + `CONNECTIONS` + ctree internal files (11) + SQL internal files (at least 1 per SQL connection)
- Maximum number of user processes should be greater than:

  `CONNECTIONS` + ctree internal (20) + Java internal (~20)
- Memory limits (`stack`, `max memory`, `virtual memory`):

  unlimited
- Core size:

  unlimited or larger than memory limits
- File size:

unlimited

Increasing the Kernel Settings for Number of Files

You may need to increase the kernel settings for number of files to properly configure and start the server. To list the current kernel limit, execute the following commands:

```
sysctl -a |grep files
kern.maxfiles 12288
kern.maxfilesperproc 12288
```

To increase the current limit, run as root:

```
launchctl limit maxfiles 32000 32000
```

This change will be reset when you reboot the system. To permanently raise this limit, create or edit the file */etc/launchd.conf* and add a line similar to the following:

```
limit maxfiles 32000 32000
```

Yosemite no longer uses */etc/launchd.conf*. Instead, it uses */etc/sysctl.conf*, which should be edited as follows:

```
kern.maxfiles=32000
kern.maxfilesperproc=32000
```

Mac Server Installation

To install the c-treeACE Server for Mac on your platform, take the following steps:

1. Make the desired directory where the c-treeACE Server is to be installed the current directory.
2. Place the c-treeACE Server CD in the drive and copy the files in the CD directories below */servers/<platform>* to the desired directory.
3. When using shared memory or message queue protocols (see `COMM_PROTOCOL` (page 162)), a directory by the name */usr/ctsrv* must exist prior to running the c-treeACE Server. The c-treeACE Server does not have to be resident in */usr/ctsrv*; however temporary files are created in this directory. Create this directory with sufficient permissions for the c-treeACE Server process to read, write, create and delete files within the directory.
4. (For versions of c-treeACE Server prior to V10.0) After installation, activate the c-treeACE Server using the **fcactvat** program. See the c-treeACE Server Activation Key Card for instructions. Some c-treeACE Server OEM vendors provide pre-activated c-treeACE Server with their applications.

Note: The c-treeACE Server for Mac OS X expects the configuration file, *ctsrvr.cfg*, to be a standard Unix text file.

2.4 c-treeACE Unix-based Servers

Supported Platforms

The following c-treeACE Unix Servers are supported:

AIX	FreeBSD
HP-UX	Linux (Intel, PPC, and Sparc)
SCO OpenServer/Unixware	Mac OS X
Solaris (Intel and SPARC)	QNX and QNX RTP

The above versions of the c-treeACE Server are installed by following the same general method and for the most part share the same hardware requirements. Items specific to a particular c-treeACE Server are discussed in "Unix Server Platform Hardware Requirements" (page 31).

Contact FairCom should you require support on other platforms. c-treeACE has been ported to dozens of platforms over the years. Generally, all that is required is a supported C compiler, and for best mutithreading support, a native pthread library. Even without native thread support, FairCom can provide a proprietary threading architecture.

c-treeACE Server Unix Installation

c-treeACE for Unix is typically distributed as a zipped tar file containing the c-treeACE Server executable, **ctreesql** or **ctsrvr**, and utility and companion programs discussed throughout this guide. Your application vendor may provide their own installation media already containing c-treeACE redistributables.

c-treeACE Server on Unix can be installed in any directory location as long as ownership and permissions permit access to resources and data.

1. Make the desired directory where the c-treeACE Server is to be installed the current directory.
2. Copy the files in the */bin/ace/* (*isam* or *sql*) directory to your desired directory.
3. Ensure you have an appropriate License Authorization File for your platform and c-treeACE configuration.

Note: Your application vendor may provide applicable c-treeACE license files.

Configuring Unix-based Systems

Systems based on Unix (including AIX, Solaris, and Linux) may have configuration requirements. In addition to configuring c-treeACE, the operating system itself may need to be configured. For example, the number of file descriptors must be large enough to accommodate the number of files c-treeACE will be access, which can be larger than the operating system's default setting.

User Limits (ulimit)

- File descriptors should be set to a number greater than:
  `FILES` + `CONNECTIONS` + ctree internal files (11) + SQL internal files (at least 1 per SQL connection)
- Maximum number of user processes should be greater than:
  `CONNECTIONS` + ctree internal (20) + Java internal (~20)
- Memory limits (`stack`, `max memory`, `virtual memory`):
  unlimited
- Core size:
  unlimited or larger than memory limits
- File size:
  unlimited

Kernel Limits

These are system-wide limits, so you must consider requirements of all processes system-wide. c-treeACE Server shared memory requirements are:

```
kernel.shmmni = 2 + shared memory CONNECTIONS
kernel.sem (argument 4) = 2 + shared memory CONNECTIONS
# increase semaphore limits for ctree shared memory.  Allows 1022 connections if no other processes
using shared memory on system
kernel.sem = 250 32000 100 1024
# number of shared memory segments.  Allows 1022 connections if no other processes using shared
memory on system
kernel.shmmni = 1024
```

For more about setting these limits, see *Shared Memory Client-Server Communication for Unix/Linux* (page 22).

Shared Memory Client-Server Communication for Unix/Linux

As of V10.0, c-treeACE for Unix supports shared memory connections. Shared memory communication between clients and servers residing on the same machine generally provides much better performance for locally running applications. Local shared memory connections are supported across the board including ISAM and SQL connections, and this includes JDBC and Windows ADO.NET Data providers.

Note: Shared memory support is not extended to Linux Kernels 2.4 or MacOS X systems as those platforms do not support the necessary interprocess mutexes.

Configuration

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

```
COMM_PROTOCOL FSHAREMM
```

c-treeACE client libraries are compiled with this featured enabled by default.

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

System Files, Permissions and Ownership

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

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

```
SHMEM_DIRECTORY <directory_name>
```

> **IMPORTANT**: `SERVER_PORT` applies to the TCP/IP protocol and overrides `SERVER_NAME` if both are used together. http://docs.faircom.com/doc/ctserver/#48603.htm (http://docs.faircom.com/doc/ctserver/#48603.htm)

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

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

```
SERVER_PORT 7000
SERVER_NAME #7000
```

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

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

```
SHMEM_PERMISSIONS <permissions>
```

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

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

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

```
SHMEM_GROUP  <group>
```

Possible errors indicating problems:

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

Shared Memory Keys

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

As of V10.4, an administrator can specify a specific shared memory key for ISAM and SQL shared memory communication protocols to ensure that keys do not match keys already in use on the system. This is specified with the following c-treeACE configuration options:

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

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

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

Client Configuration

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

Errors with Shared Memory Protocol

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

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

Adjusting System Limits

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

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

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

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

The following error can be reported as well:

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

These are typically kernel configurations. c-treeACE requires (`2 + # shared memory CONNECTIONS`) shared memory segments (`SHMMNI`) and semaphores (`SEMMNI`).

The **ipcs** command displays current limits:

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

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

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

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

Run this command to then enable support:

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

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

Usage

To take advantage of this feature, check the following:

1. Shut down your c-treeACE Server and add the following keyword to your *ctsrvr.cfg* file:
   `COMM_PROTOCOL FSHAREMM`
2. Restart c-treeACE.
3. Execute any c-treeACE utility you've linked with V9.5 or later of c-treeACE you have on the same machine as the c-treeACE Server process. Even if you are linked with a c-tree TCP/IP library, it will automatically detect if you are running on the same machine and try to connect via shared memory. This way you don't need multiple versions of your application and utilities.
4. You can monitor your connections by listing the clients from the **ctadmn** command-line utility on Linux, or by using the c-treeACE Monitor program from Windows (it is shown in the **Comm Info** column on the far right).

Usage Notes

- Unix and Windows client libraries are built with shared-memory support by default.
- When c-treeACE detects a request to connect from the same machine as the client, it first attempts to connect using shared memory. If that succeeds, the connection uses the shared-memory protocol. Otherwise, the connection is made using TCP/IP.
- 32-bit clients can connect to 64-bit servers (and vice versa).
- By default, a client application must belong to the server owner's primary group to use shared memory. This is configurable with the `SHMEM_GROUP` keyword.

- Shared memory uses a Unix domain (file system) socket for transferring data between the client and server. The Unix domain socket exists as a file named `/tmp/ctreedbs/<servername>.logon`.

  `COMPATIBILITY SHMEM_PIPE` can be specified to use an earlier approach of named pipes.

- A Unix/Linux server using shared-memory communications will create a directory */tmp/ctreedbs*. If this directory already exists (for example, if a different user had started the server, even from a previous run) and the server does not have write permission to this directory, startup will fail, most likely reporting a **DSRV_ERR** error (509, duplicate server), even if no other server is currently running.
- Prior to V10.3, pthread mutex shared-memory support is expected for Unix systems. However, if a client and the server are compiled with incompatible options (for example, the client uses System V semaphores but the server uses pthread mutexes) the connection attempt will fail and c-treeACE will log one of the following messages to *CTSTATUS.FCS*:

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

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

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

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

System Tools

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

Solaris Considerations

Orphan shared memory segments and semaphores contribute to the system limit. Their presence can affect the number of user connections you will be able to achieve. There is no way of ensuring these are removed if the server process exits unexpectedly, so they must be removed manually. To remove them, follow these procedures:

1. List the **shared memory segments** and the number of attached processes (NATTACH):

   `ipcs -m -o`

   This shows all the shared memory segments on the system. Ones with no attached processes might be safe to remove.

   Note: Be careful not to delete the shared memory segments created by other processes. Deleting a shared memory segment that is in use does not cause an immediate problem for c-tree. This is similar to deleting a file that is open: any process using it can keep using it, but no one can start using it.

2. Use the `ipcrm` command to remove unwanted shared memory segments and semaphores. Consult the manpage on your system for specific details about using this command.
3. List the **interprocess semaphores** using the `ipcs -s` command.

   For Solaris 5.10, see *List Interprocess Semaphores on Solaris 5.10* below.

   Solaris does not provide a way to see which interprocess semaphores belong to a specific process. To get an idea of whether a semaphore might be in use, use `ipcs -s -t` to see the

time of last use. You may be able to infer which semaphores belong to c-tree by the time of last use (especially if you can shut down c-tree cleanly and observe the change in semaphores).

4. Delete any unwanted interprocess semaphores.

Note: Deleting an interprocess semaphore if it is actively in use by c-tree causes the connection to immediately fail. Deleting an interprocess semaphore that is in use by another process may cause that process to fail.

Once you have removed the items shown above, you can start the server and connect to it.

Raising the Limits on Solaris 5.10 and Later

The following parameters are useful for managing resources:

- *max-shm-ids* - Maximum number of shared memory segments on a system
- *max-sem-ids* - Maximum semaphore IDs for a project.
- *max-sem-nsems* - Maximum number of semaphores allowed per semaphore set.

To temporarily raise the limits set for these parameters, run the following as root, where `SHELL_PID` is the PID of the shell that will be starting the c-tree server process:

```
prctl -n project.max-shm-ids -r -v 1024 -i project default
prctl -n project.max-sem-ids -r -v 1024 -i project default
prctl -n process.max-sem-nsems -r -v 1024 SHELL_PID
```

You can then start c-treeACE Server from the shell with `PID = $SHELL_PID` and connect users.

The above settings will be reset to the default when the machine is rebooted. (Also, remember that `max-sem-nsems` is only increased for that shell process and its children).

To make these changes permanent so they are effective after every reboot, execute the following command:

```
/usr/sbin/projmod -sK "project.max-shm-ids=(privileged,1024,deny)" default
```

After changing the shared memory parameters, you may need to delete orphaned shared memory segments as described above.

List Interprocess Semaphores on Solaris 5.10

Solaris 5.10 (and later) has a different method for capturing the current number of semaphores:

1. Run the following command:
   ```
   prctl -n project.max-shm-ids $$
   ```
   Example output:
   ```
   /export/home/fctech$ prctl -n project.max-shm-ids $$
   process: 3636: -tcsh
   NAME    PRIVILEGE       VALUE    FLAG   ACTION       RECIPIENT
   project.max-shm-ids
           privileged        128       -   deny                 -
           system          16.8M     max   deny
   ```
2. Run the following command:
   ```
   prctl -n process.max-sem-nsems $$
   ```

Example output:

```
/export/home/fctech$ prctl -n process.max-sem-nsems $$
process: 3636: -tcsh
NAME    PRIVILEGE       VALUE    FLAG   ACTION       RECIPIENT
process.max-sem-nsems
        privileged        512       -   deny                 -
        system          32.8K     max   deny                 -
```

c-treeACE Server - Unix version logs message when shared memory can’t create semaphore or segment

n Unix systems, c-treeACE Server V11 and later logs a message to *CTSTATUS.FCS* when it fails to create a system semaphore or shared memory segment due to a resource limit. This helps the server administrator understand the cause of the error. Example message:

```
Fri Aug 22 15:14:44 2014
 - User# 00013  FSHAREMM: Failed to create system semaphore: check system semaphore limits such
as SEMMNI
Fri Aug 22 15:14:44 2014
 - User# 00013  NewUser: Unable to create an instance of a named pipe
```

Other Possible Shared Memory Messages

c-treeACE ISAM and SQL ports are independent of each other. In general, there is a shared memory connection protocol enabled for each, in addition to TCP/IP ports. Keep in mind that's a total of four (4) connection protocols, with configuration options for each.

Possible SQL shared memory connection errors can appear such as the following. Analyze and correct these as you would for the ISAM errors previously mentioned as the same parameters should be examined.

```
- User# 00012   sqlshmlisten: shared memory protocol initialization failed: -1
- User# 00012   sqlshmlisten: shared memory protocol accept failed: -2
- User# 00012   sqlshmlisten: Failed to get shared memory environment: XX
```

Named pipe creation failed with error ERROR_PIPE_BUSY. The operation will be retried

```
 - User# 00012  sqlshmlisten: shared memory protocol listen failed: -4
```

Failed permissions on the temporary directory needed.

```
- User# 00012  SQLSHAREMM: Failed to open /tmp/ctreedbs/CTSQL_6597 for shared memory: 13
```

Unix shared memory protocol not freeing shared memory segments (different client and server user accounts)

When clients used the shared memory protocol on Unix systems and the client and server processes were run under different user accounts, shared memory segments could be left behind after the connections were closed. The **ipcs** *-m* listing showed shared memory segments with no processes attached. In V10.3 and later, the logic has been modified to correct this.

Note: These changes add a field to the client and server logon data structures to pass the user ID between the client and server processes, resulting in the following compatibility considerations:

An old client can connect to a new server and it will behave as it did before these changes.

A new client cannot connect to an old server using shared memory. It will receive **error 133** if the server is configured to only use shared memory. It will connect with TCP/IP if the server is using both shared memory and TCP/IP. The old server will log the following message to *CTSTATUS.FCS*:

```
FSHAREMM: The client's shared memory version (2) is not compatible with the server's shared memory version (1)
```

At startup, the c-treeACE Server now logs messages to *CTSTATUS.FCS* to indicate the shared memory directory used for logon purposes and the shared memory protocol version that it is using:

```
FSHAREMM: SHMEM_DIRECTORY=/tmp/ctreedbs/
FSHAREMM: Protocol version=2
```

Specify Shared Memory Keys on Unix

When more than one c-treeACE Server was run on a Unix system, the shared memory keys used by different servers could have the same value, which prevented connections to the servers. In addition, it was possible for unrelated applications to collide with default keys generated by c-treeACE servers.

To address this key collision, it is now possible for an administrator to specify specific shared memory keys for ISAM and SQL shared memory communication protocols ensuring the keys do not match existing keys already in use on the system.

New c-treeACE Server configuration options are available to directly specify a shared memory key. SQL and ISAM each require separate shared memory support

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

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

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

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

Compatibility Notes:

When these configuration options are *not* used, c-treeACE Server uses the old method of assigning shared memory keys so its shared memory communication protocol is compatible with old clients. c-treeACE Server now writes the shared memory key to the shared memory resource file and new clients know to read the shared memory key from this file. If a new client finds no shared memory key in the file, it uses the old method to assign a shared memory key so it is compatible with an old server.

The shared memory resource file is named */tmp/ctreedbs/<server_name>* for ISAM, and */tmp/ctreedbs/CTSQL_<sql_port>* for SQL, where */tmp/ctreedbs* is the default shared memory directory. It can be changed using the `SHMEM_DIRECTORY` configuration option.

An old client will not be able to connect to a new server using shared memory if the server uses the `SHMEM_KEY_ISAM` or `SHMEM_KEY_SQL` configuration option to specify a shared memory key that differs from the shared memory key that the old method would generate.

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

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

The following shared memory protocol changes were enacted:

- Now c-treeACE Server uses a Unix domain socket instead of a pair of named pipes for the initial communication when a client connects to the server. c-treeACE Server still creates the named pipes, and when a client connects, the server waits for the client to write to either the socket or the named pipe. In this way, the server is able to support both clients that use the new method and those that use the original method.
- The c-treeACE Server configuration option `COMPATIBILITY SHMEM_PIPE` can be used to restore the original behavior of only using named pipes. We expect this keyword to be used only in an unexpected situation in which the new option is not working as well as the original option.
- c-treeACE clients (both ISAM and SQL) now use the Unix domain socket method when connecting using the shared memory protocol if the server indicates that it supports it. If not, the clients use the original method.
- A c-treeACE client library can be compiled with `#define NO_ctFeatUNIX_SHMEMsocket` to force the client to use the original method only.

System Group Assignment of Unix/Linux Shared Memory resources

On Unix/Linux systems, a user can belong to more than one group of which one group is the primary group, and all other groups are secondary groups. When the `SHMEM_PERMISSIONS` option is used to only enable user and group permissions on shared memory resources, the resources created for shared memory connections (files, semaphores, shared memory regions) are assigned with the user's current primary group by default.

To address this situation, a new configuration option, `SHMEM_GROUP`, has been added preventing a user account that shares a secondary group with the user account under which the c-treeACE Server process is running failing to connect using with shared memory.

This option causes c-treeACE Server to assign group membership to the specified group. This option applies to the resources for both the ISAM and the SQL shared memory protocol.

As an example, consider two user accounts:

- `user1` - belongs to groups `group1` and `group2`
- `user2` - belongs to group `group2`

If the `user1` account runs c-treeACE Server with `SHMEM_PERMISSIONS 660` in *ctsrvr.cfg*, a client program run by the `user2` account will fail to connect using shared memory.

To allow the client program run by `user2` to connect, add the following configuration option to *ctsrvr.cfg* and restart c-treeACE Server:

```
SHMEM_GROUP group2
```

This causes the shared memory resources to be assigned to group `group2`, which allows the `user2` client program to connect.

Unix Server Platform Hardware Requirements

The requirements for the c-treeACE Server on each listed operating system follow:

Minimum Hardware Requirements

The minimum hard drive space required by c-treeACE Server for Unix is:

The size of the c-treeACE Server executable
+ the amount specified by the `LOG_SPACE` keyword (10 MB default)
+ 1MB for c-treeACE Server status logs
+ size of pre-compiled c-treeACE Server utilities
+ the size of the data (*.dat*) and index (*.idx*) files

Hewlett Packard HP-UX

The c-treeACE Server for HP-UX requires a minimum of 2MB RAM. The HP-UX 11 operating system, and above, supports standard c-treeACE files up to 4 GB in size and allows huge files. Earlier versions support 2GB file sizes and requires segmented files to support larger files.

For proper operations of the c-treeACE Server under various loads, FairCom recommends adjusting the following kernel parameters of the HP/UX 11 system, using the **sam** utility:

- Increase maximum per-process stack memory size (*maxssiz*) from the default of 8 MB to 128 MB.
- Increase maximum per-process data memory size (*maxdsiz*) from the default of 64 MB to 256 MB.
- Consider increasing the number of threads per process if connecting a large number of clients. The default for older releases of the OS is relatively low (64 maximum threads per process).

- Either increase the default number of file handles from 60 to 256 by using **sam** or, prior to starting the c-treeACE SQL process, issue `limit descriptors 256` to increase the number of file descriptors used by that process only.

IBM AIX

The c-treeACE Server for IBM AIX requires a minimum of 2MB RAM. There are specific versions for AIX v5.1 and above and these all support both native and proprietary threading. See "Native Threads" (page 32) for additional information. The AIX v5.1 (and above) operating systems can be configured to support standard c-treeACE files up to 4 GB in size and allow huge files.

Linux

The c-treeACE Server for Linux requires a Pentium 133, Sparc, or PPC CPU and a minimum of 2MB RAM. Linux versions using kernel 2.4 and above support Standard c-treeACE files up to 4 GB in size and allows huge files. Earlier versions support 2GB file sizes and requires segmented files to support larger files.

QNX and QNX RTP

The c-treeACE Server supports QNX Software's proprietary communication protocol and TCP/IP. Specific requirements include a Pentium 133 or greater CPU and a minimum of 2MB RAM. This operating system supports only 2GB file sizes and requires segmented files to support larger files.

SCO OpenServer / Unixware

The c-treeACE Server for SCO OpenServer and UnixWare requires a Pentium 133 or greater CPU and a minimum 2MB RAM. This operating system supports only 2GB file sizes and requires segmented files to support larger files.

Solaris - SPARC and Intel

The c-treeACE Server for Solaris requires a minimum of 2MB RAM. The Solaris 2.8 and above operating systems supports standard c-treeACE files up to 4 GB in size and allows huge files.

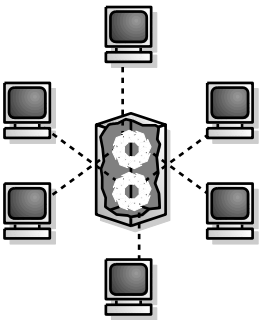
Native Threads

FairCom has enjoyed multi-threaded support from nearly our inception and has extensive engineering experience in supporting various threading architectures that have appeared through the years. POSIX pthread support is considered the industry standard in recent times, and we highly encourage taking advantage of this support when at all possible. By default, all c-treeACE servers include pthread support when available on a chosen platform.

For platforms not supporting a native threading technology, FairCom supports a proprietary threading library. FairCom recommends native thread support when available as performance enhancements are typical.

Note: The Native Thread Server supports only the TCP/IP communications protocol.

2.5 Multiple c-treeACE Servers on One Machine



It possible to install and run multiple instances of the c-treeACE Server on the same machine.

To run multiple c-treeACE instances, each instance must be individually licensed, have a unique name (communications ports), and have a unique set of server and data files.

Important! Separate c-treeACE instances cannot share files.

For each c-treeACE instance, you must:

1. Install c-treeACE into a unique directory.
2. License each c-treeACE instance. Every instance of c-treeACE requires a separate license.
3. Specify a unique Server Name in the configuration file, *ctsrvr.cfg*, as described in "Configuring the c-treeACE Server" (page 153). (Don't forget unique SQL port numbers.)
4. Configure each instance to have a unique Service Name if running as a Windows service.(Described in "Configuring the c-treeACE Server" (page 153).)

2.6 TCP/IP Broadcast Support

It is possible for your vendor to create client applications that listen for an available c-treeACE Server without knowing the Server Name in advance. A c-treeACE Server can be configured to broadcast its Server Name and IP address over a TCP/IP port. With this method, it is possible for a client to detect the various c-treeACE Servers operating on the network and obtain their Server Names, including IP addresses. Your vendor will notify you if and when you should use these settings and what values should be used.

Three server keywords support the broadcast feature: `BROADCAST_PORT`, `BROADCAST_INTERVAL`, and `BROADCAST_DATA`. See the examples in "c-treeACE Configuration Options" (page 160).

- `BROADCAST_PORT` specifies the TCP/IP port used for the broadcast. The default value is 0, which means the broadcast is off. If DEFAULT is specified, this means that the broadcast is on and the default port is used, which is 5595. Any valid four-byte integer greater than 5000 that is not in use by another process may be specified. This should NOT be the port for the c-treeACE Server, which is displayed at startup and is based on the Server Name. See the examples below.
- `BROADCAST_INTERVAL` determines the number of seconds between broadcasts. The default is 10 seconds, otherwise the token should be a number. If the number is negative, each broadcast is also sent to the c-treeACE Server standard output. To prevent unreasonable

values, the maximum value allowed is currently set to 86,400 seconds, which is once per day.

- `BROADCAST_DATA` specifies a token to be broadcast following the Server Name. The token must not contain spaces. The Server Name will be followed by a vertical bar character, '|', which is followed by the token. There is no default token.

Using the following sample keywords and assuming the host IP address was 127.0.0.1, the c-treeACE Server broadcasts "SAMPLE | 127.0.0.1 | 5451| FAIRCOM_SERVER" on port 6329 every 90 seconds:

```
SERVER_NAME          SAMPLE
BROADCAST_PORT       6329
BROADCAST_INTERVAL   90
BROADCAST_DATA       FAIRCOM_SERVER
```

2.7 Heterogeneous Server Network Support

The c-treeACE Server automatically provides sophisticated network support allowing dissimilar machines to share data across the same network. The FairCom term for this type of logic is "Netformat". Netformat logic automatically controls all aspects of data byte ordering (big endian/little endian). The server process defines the ordering of the data (High/Low (big endian) or Low/High (little endian)) while the client process dictates the alignment of the file.

Example

To further illustrate the power of the Netformat logic, review the following network scenario:

- Server: c-tree IBM POWER7 TCP/IP Server
- Clients: Intel based Windows word aligned client application.

All data files will be stored in a High/Low (Most significant byte/Least significant byte) format used by the IBM POWER7 CPU. Files created by the Intel Windows application will be word aligned (the default with Microsoft Visual Studio compiler). The applications will all be able to share the same files (assuming the application developer has aligned all numeric fields on at least a 2-byte boundary for this example - a good C programming practice).

3. c-treeACE Server Basic Operations

Once the c-treeACE Server is installed on the operating system, it is ready to be used. Starting and stopping the c-treeACE Server are basic Administrator responsibilities, therefore this chapter is required reading.

3.1 Starting c-treeACE the First Time

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

1. If the vendor has supplied an (optional) encrypted 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 *c-treeACE Configuration Options* (page 160) for more information.
2. See if there is a c-treeACE Server configuration file, *ctsrvr.cfg*. If so, verify the file contents, change it if necessary, and prepare it to run when the c-treeACE Server starts. See *Configuring the c-treeACE Server* (page 153) for details.
3. Change the Administrator's password to protect future access to the c-treeACE Server and access to Administrator utilities. Use the Administrator Utility described in *c-treeACE Server Administrator Utility* (page 56).

   Note: Initially, the c-treeACE 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 *c-treeACE Server Administrator Utility* (page 56).
4. Set up initial User IDs so users can log on to the c-treeACE Server. Use the Administrator Utility, described in *c-treeACE Server Administrator Utility* (page 56).

3.2 Starting c-treeACE

The following is a general discussion of the process used to start a c-treeACE Server. In most environments, **ctreesql** is the name of the c-treeACE SQL Server executable (**ctsrvr** is the name of the ISAM-only c-treeACE Server).

1. Ensure **ctreesql** (or **ctsrvr**) is in the base directory for database operations. See *c-treeACE Server Installation* (page 7).
2. If reconfiguring the c-treeACE Server, use a text editor to create a configuration file, *ctsrvr.cfg*. See *Configuring the c-treeACE Server* (page 153).
3. If adding (or changing) a configuration file, make sure it is in the same directory as the c-treeACE Server, is optionally set with the *FCSRVR_CFG* environment variable, or is listed on the command line (*CTSRVR_CFG <file>*).

Note: If the configuration file is not found by the c-treeACE Server, the server will not use the customized configuration file but will begin operation using default configuration settings. Check the installation instructions for your platform in *c-treeACE Server Installation* (page 7) for any exceptions.

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

Note: The c-treeACE Server name may have a file extension - see the platform-specific information in *c-treeACE Server Installation* (page 7) for details.

Note: By default, no password is required to start the c-treeACE Server, therefore an automated process, such as a batch, script, or cron process, may start the c-treeACE Server.

Every time the c-treeACE 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* (page 97) for details.

In most Unix environments, FairCom recommends Administrators run the c-treeACE Server in background to decrease the chance of unwittingly terminating it. For example:

```
# ctreesql &
```

or

```
# ctsrvr &
```

The Unix “no hang up” option may also be used to keep the c-treeACE Server from being terminated if the user starting the c-treeACE Server logs off the system. For example:

```
# nohup ctreesql &
```

or

```
# nohup ctsrvr &
```

Start Up Errors

The c-treeACE Server verifies database integrity and the operation of its own components at startup. If any problems are detected, the c-treeACE Server places error messages in the c-treeACE Server Status Log, *CTSTATUS.FCS*, and displays them on the system console. In extreme cases, the c-treeACE Server halts operation. Several kinds of errors can occur at startup.

Some of these errors, and the appropriate reaction to each, are as follows:

Error	Explanation and Reaction to Error
12	A file required during automatic recovery cannot be located. Either it was removed after the server failed or the physical media (e.g. disk drive) was damaged during a failure. To recover from this problem, reload the last complete system backup. See "Maintaining Database Integrity" (page 92) for more information.
46	File number overflow. Too many files opened. Verify the configuration to ensure the proper file allocations and limits. Verify the applications are opening the appropriate files in the appropriate manner.

Error	Explanation and Reaction to Error
96	A log file required for recovery is not available. To recover from this problem, reload the last complete system backup. See "Maintaining Database Integrity" (page 92) for more information.
143	Communication handler not installed. Be sure the network drivers are loaded or the shared memory directory (*/usr/ctsrv*) for Unix platforms has been created.
173 174	Can also occur if the server is unable to create a socket for the client connection. Socket creation will fail if the available file descriptors for the server process have been used (if the server has many files open, for example). Check the per-process file descriptor limit on your system. This limit must be large enough to accommodate the number of files you wish to open (the `FILES` setting in the server configuration), plus the number of clients to connect to the server at a time (the `CONNECTIONS` or `USERS` setting in the server configuration).
509	Another copy of the particular c-treeACE Server you attempted to start is already running. Each installed copy of the c-treeACE Server must have its own license and a unique serial number. If you do not have enough c-treeACE Server licenses, please contact your c-treeACE Server provider.

Note: Utility error messages covered in this manual refer to messages a c-treeACE Server sends to a program connected to it. Although we list error numbers with brief explanations of each error, it is important to understand how errors are treated, including messages sent to users. It is the responsibility of the client application programs receiving the error messages to properly capture and display the errors.

Note: The error messages associated with specific error numbers for the c-treeACE SQL Server are found in the *dherrors* file which is stored in the *lib* sub-directory below the `LOCAL_DIRECTORY` (if this keyword is defined in the server configuration file), `SERVER_DIRECTORY` (if this deprecated keyword is defined in the server configuration file), or the directory where the server is installed.

Errors Ignored When IP Address Return for Host System Fails

In V11 and later, c-treeACE Server is able to ignore some errors getting the IP addresses for the host system when it starts. Sometimes c-treeACE Server failed to start with error **891** (most commonly seen on Unix systems). The following message was logged to *CTSTATUS.FCS*:

```
- User# 00001  Failed to get IP address for host 'hostname': gethostbyname() returned error code
1
```

This error occurred when the system's host name (as shown by the **hostname** or **uname** *-a* commands) was not present in the */etc/hosts* file and could not be resolved by DNS.

c-treeACE Server now ignores this error except in situations that require getting the IP addresses for the host system (when using the "local connections only" and "node-based licensing" options).

Launching c-treeACE Server companion executable

The `SIGNAL_READY` keyword provides the ability to launch an executable when the c-treeACE Server comes up. This keyword takes as its argument the name of an executable to be launched when the c-treeACE Server is ready (i.e., automatic recovery is completed). See `SIGNAL_READY` in *c-treeACE Configuration Options* (page 160) for additional information.

This option allows applications that require the Server to launch automatically. Some examples including an auditing application, subordinate controlling interfaces (bar code readers, etc.), or a client running on the same machine as the Server.

3.3 Stopping the c-treeACE Server

Only a user in the ADMIN group can stop the c-treeACE Server. The server can be stopped using the Windows Close icon or menu item, using the **ctstop** or **ctadmn** utilities, or by an application using the **StopServer()** function.

To stop the c-treeACE Server with the Close button in the upper-right corner of the application window, or with the Close item in the File menu, just click either object, as is common with any other Windows application. An ADMIN group User ID and password is required to complete the close operation.

To stop the c-treeACE Server with the module **ctstop**, a special client application:

1. Start this program like any other.
2. The stop module asks for four things:
   a. The ADMIN user ID, which must be ADMIN or a member of the ADMIN group.
   b. The user password, which is necessary for continuing with the procedure.
   c. The current name of the c-treeACE Server, if an alternative to the default name was given in the configuration file (see the keyword `SERVER_NAME` in "Basic Configuration Options") to specify which c-treeACE Server to stop.
   d. The delay time (if any) before shutting down the c-treeACE Server. If a greater-than-zero delay is specified, the c-treeACE Server will not accept any new users or transactions. Logon attempts during the delay time specified will fail with error **SHUT_ERR** (150), which means, “The Server is shutting down”. New transactions cannot be started while waiting to shut down. They will return **SHUT_ERR** (150) or **SGON_ERR** (162), “Server gone”, depending on how far the shutdown process has gone.

Tip: This can be given on a single line as: >

```
ctstop ADMIN ADMIN FAIRCOMS
```

The c-treeACE Server may also be stopped by an application program, as long as it supplies an ADMIN group User ID and password, using **StopServer()** discussed in the *c-treeACE Programmer’s Reference Guide*, (distributed only to c-treeACE developers).

During c-treeACE 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-treeACE Server shutdown. If these loops are entered, the c-treeACE 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-treeACE Server Administrator to monitor the shutdown, and avoid an incorrect assumption about whether the c-treeACE Server is making progress or has hung during shutdown. After the c-treeACE Server shuts down, it sends a message saying c-treeACE 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.

Launching Server companion upon shutdown

The `SIGNAL_DOWN` keyword provides the ability to launch a customer executable when the c-treeACE Server comes down. This keyword takes as its argument the name of an executable that will be launched when the c-treeACE Server has been successfully terminated. See `SIGNAL_DOWN` in "<FC_PRPD_ACE> Configuration Options" (page 160) for additional information.

This option could be used to launch a backup utility, to re-launch the server, or to execute a batch/shell script to perform actions that can only be performed while the server is inoperative.

3.4 Server Operational Errors

The c-treeACE Server performs rigorous error checking and logging during the course of everyday operation. Because of the depth of error checking that is performed, warnings and error messages are logged in even the most benign situations.

Windows Resource Error (1450) Configurable Retry Logic

When the Windows kernel has allocated all of its paged-pool memory, it will not be able to perform many tasks and instead returns a STATUS_INSUFFICIENT_RESOURCES (0xC000009A) message. This is a restriction of 32-bit addressing (only 2GB addressable within the kernel), regardless of the amount of memory available in the system.

Microsoft Support Knowledgebase regarding Error 1450 http://support.microsoft.com/kb/142719

When the c-treeACE Server configuration option `IO_ERROR_BLOCK_SIZE` option is specified in the c-treeACE Server configuration file, a read or write operation that fails with Windows system error 1450 (ERROR_NO_SYSTEM_RESOURCES) is retried in blocks of the specified size. If any one of those read or write operations fails, the c-treeACE Server fails the read or write operation.

The c-treeACE Server supports two additional configuration options that permit additional disk read/write retries and a sleep interval between retries.

`IO_ERROR_BLOCK_RETRY <retries>` specifies the maximum number of failed IO_ERROR_BLOCK_SIZE-sized I/O operations that must occur before the I/O operation is considered to have failed. If the IO_ERROR_BLOCK_SIZE-sized I/O operations that are being attempted for a particular I/O operation fail more than *<retries>* times, the c-treeACE Server writes a **READ_ERR** (36) or **WRITE_ERR** (37) message to *CTSTATUS.FCS* and considers the I/O operation to have failed.

A value of -1 signifies infinite retries. The default is 0, which means that the I/O operation is tried only once in IO_ERROR_BLOCK_SIZE-sized blocks, and if any of these I/O operations fails, the entire I/O operation is considered to have failed. As another example, if IO_ERROR_BLOCK_RETRY is set to 20 and IO_ERROR_BLOCK_SIZE is set to 65536, if a 327680-byte write is retried as 5 65536-byte write operations, then the I/O operation fails if there are 20 failures to perform those 5 write operations.

`IO_ERROR_BLOCK_SLEEP <time>` specifies a time in milliseconds between retry attempts. The default is zero, which means that retries are attempted immediately.

SNAPSHOT Monitoring of Failed Retires

To permit monitoring the number of I/O error 1450 retries that have occurred, a counter has been added to the system snapshot structure. The *sctioblkretry* field of the *ctGSMS* structure is defined as an unsigned long integer that stores the total number of I/O error 1450 retries that have occurred since the c-treeACE Server started. The snapshot log file *SNAPSHOT.FCS* displays the I/O error 1450 retry counter value with a description of "I/O ERR(1450) automatic retries:". The system snapshot structure version has been changed from 9 to 10 to note the presence of this new field in the structure and the statistics monitoring utility, **ctstat**, and **ctsnpr** utilities have been updated to properly handle the presence of this field in the system snapshot structure and snapshot log.

I/O Block Sizes With Windows Systems

When the `IO_ERROR_BLOCK_SIZE` configuration option is specified in the c-treeACE Server configuration file, if a disk read or write operation fails with system error 1450 (Insufficient system resources exist to complete the requested service), the server retries the I/O operation using the specified block size. If the retried I/O operation also fails with error 1450 (or if an disk I/O operation fails with error 1450 and `IO_ERROR_BLOCK_SIZE` is not specified in the server configuration file), c-treeACE now logs the following message to *CTSTATUS.FCS*:

```
<op_code>: loc <location> file <filename> offset <offset> iosize <iosize> syserr <errcod>
```

where:

- *<op_code>* is **READ_ERR** (36, indicating that a read operation failed) or **WRITE_ERR** (37, indicating that a write operation failed)
- *<location>* is the location in the code where the I/O operation failed
- *<filename>* is the name of the file for which the I/O operation was requested
- *<offset>* is the offset of the failed I/O operation
- *<iosize>* is the size of the failed I/O operation
- *<errcod>* is the system error code for the failed I/O operation

An internal write call was modified to ensure that when the `IO_ERROR_BLOCK_SIZE` configuration option is used, a retried I/O operation is done at the original offset for the I/O operation and that *sysiocod* is reset to zero before retrying the I/O operation.

Communications Errors (127/128)

When a communication error such as **VDP_ERROR** (127 or 128) occurs, the c-treeACE Server logs an entry in the Server Status Log *CTSTATUS.FCS*. This is not a serious situation unless the client application is also getting errors such as 127, 128, or similar errors.

The context of the communications error is that a server thread gets a notification that a message is available, but when the server performs a read, nothing is returned.

This can be caused by:

- Physical network problems

- An overworked network transport layer that is timing out and doing retries
- Clients exit without calling **CloseISAM()** or **StopUser()**, or end users that turn their machines off without properly logging out of the application. This category includes client application crashes.

To ensure the errors are not serious, try to reconcile the communications errors in the log with the client events that triggered them. Since these errors do not usually happen frequently and user names are provided, it should be easy to determine which event caused this situation.

To avoid there errors, ensure the c-treeACE Server's host machine is not burdened beyond its capacity. Using a more powerful machine or limiting the number and types of applications on a machine can improve performance and limit errors at the communication level. Also, ensure no specific application is over-using resources on the host machine. If appropriate in the server's operating environment, increasing the priority of the c-treeACE Server can eliminate or reduce communications errors. This should be done cautiously as it will affect other applications running on the same machine.

The error messages in the Server Status Log can be turned off, but unless they are an inconvenience, this is NOT recommended. The messages serve as a good health check on the state of your network and may be an early warning of more serious network and system problems. To disable the messages, add

```
CTSTATUS_MASK  VDP_ERROR
```

to the *ctsrvr.cfg* file and restart the c-treeACE Server.

8770

The 8770 error occurs when the server attempts to remove an internal unique file ID from a list as a file is closed, but that file ID is not on the list. This might be caused by an application opening different files with the same internal file ID. This would typically be the case when a file is copied and both files are then opened; therefore they both have the same file ID. To avoid these errors do not copy server-controlled files. If the 8770 error occurred after another more serious error, the 8770 error can be safely ignored. If it recurs, contact your application developer for assistance.

3.5 Server Memory Calculations

The c-treeACE Server startup memory requirements can be reasonably approximated with the following equation:

```
Base line Memory =
    Server EXE size + 1 MB +
    Communications Library size (if applicable) +
    Data cache (DAT_MEMORY) +
    Index buffer (IDX_MEMORY) +
    (900 bytes * Number of files (FILES)) +
    (325 bytes * Maximum number of connections (CONNECTIONS)) +
    (4 bytes * Maximum number of connections (CONNECTIONS)
             * Maximum number of connections (CONNECTIONS)) +
    (16000 bytes * Number of actual connections) +
    (256 bytes per connection per file actually opened))
```

Note the following points:

- DAT_MEMORY and IDX_MEMORY defaults vary based on the type of server (Standard vs. SQL) and the PAGE_SIZE. See "Basic Keywords" (page 161) for details.
- FILES defaults to 1000.
- CONNECTIONS default to 128.
- IDX_MEMORY is the MAX of:
  - IDX_MEMORY or
  - 3 * CONNECTIONS * (PAGE_SIZE + 400), where PAGE_SIZE defaults to 8K.

The following locking/transaction processing related items should be considered when approximating the c-treeACE Server dynamic memory requirements:

- Each record locked consumes 24 bytes.

For transaction processing files only:

- Each data record written consumes (record length + 42) bytes.
- Each index operation consumes (key length + 42) bytes.

Note: Some operating systems offer virtual RAM (VRAM) which swaps data from memory to the hard drive. VRAM is usually automatically invoked if RAM gets full. Since data is being moved to and from the hard drive by VRAM, applications will often slow. If your system suddenly slows, examine this possible cause with your system administrator.

3.6 Stack Traces in Case of Critical Error

A diagnostic feature is available that provides a stack trace showing calls for all threads. This is performed automatically when a fatal error occurs.

This feature redirects output to the file *pstack<server_pid>_<sernum>.log*, where *<server_pid>* is the process ID of the c-treeACE Server process and *<sernum>* is a serial number maintained by the server to ensure unique log names. The server also writes a message to its status log, *CTSTATUS.FCS*, indicating that a process stack trace was dumped. For example, the following message in *CTSTATUS.FCS* refers to the file *pstack_454_01.log*:

```
Dumped stack for server process 454, log=1, loc=73, rc=0
```

Please contact your vendor immediately if you encounter a c-treeACE crash and be prepared to supply these stack dumps. This enables FairCom engineers to quickly pinpoint critical errors and enable fast response times for corrected servers.

Note: To receive function times, you will need shared libraries (Unix/Linux) or DLL (Windows) with symbols. Be sure to run the **pstack** command from the local directory where the c-treeACE Server library resides. If it is not run from that directory, the symbol-enabled library will not be loaded and the system function calls will appear as question marks in the call stack.

Unix Operating Systems

For servers running on Unix operating systems, the implementation invokes the **pstack** utility.

Note: If **pstack** does not exist on your system, no trace will be created.

If your system's process stack trace creation utility is named something other than **pstack**, you can create a shell script named **pstack** to run your utility. For example, if the utility on your system is named **dumpstack**, create a shell script named **pstack** with the following two lines:

```
#!/bin/csh

dumpstack $1
```

Remember to give your shell script execute permission (for example, **chmod +x pstack**) and put it in a directory that is in the path.

IBM AIX Operating Systems

The server uses the AIX **procstack** utility to log the stack trace.

Microsoft Windows Operating Systems

This feature requires an external DLL, *dbghelp.dll*, which is part of the Windows installation, and is dynamically loaded at stack dump time. It also requires that this DLL exports the function **MiniDumpWriteDump()** which may not be the case with older versions of this DLL. In the case where the DLL cannot be found or it does not contain the function, the stack dump fails and a message is logged in *CTSTATUS.FCS* without any other consequence.

It is suggested to enable Dr. Watson on the Windows system of interest when attempting to generate a dump file with information needed to trace a continuing server stack dump. Visual Studio 7.0 or greater is required to inspect the file for Windows stack dumps.

Security Note: In-flight data will be captured at the time these core files are generated. Consider the case of data and indexes residing in the memory caches of the server process space. This will be unencrypted data even if it was encrypted on disk. Please consult with your local risk assessment organization concerning any privacy or security issues before forwarding this information to FairCom.

4. c-treeACE Access Configuration

One of the main responsibilities of a c-treeACE Server Administrator is to establish and maintain access to the c-treeACE Server. Although reviewing this chapter is not required for operating the c-treeACE Server, FairCom recommends Administrators consider the following features.

Access to the c-treeACE Server can be controlled in four basic ways:

User access restrictions	Requiring User ID and/or User password to access the c-treeACE Server.
File access restrictions	Requiring file password to access a file.
File operation permissions	Controlling which specific operations (e.g., read, write, delete) a given class of users can perform on a particular file they have accessed.
Group-based restrictions	Defining groupings, then assigning users to given groups and giving appropriate file permissions only to members of a specific group.

The details of access and security control through user, file, and group information are covered in this section. Basic concepts needed to understand security operations are covered first. Descriptions of the Administrator Utility used to enter security information for the c-treeACE Server and monitor users while they are connected to the c-treeACE Server follow.

Note: The controls discussed here are those available to the Administrator.Applications can also be programmed to allow certain security controls (e.g., change file passwords) to users who have appropriate access to the c-treeACE Server, using available security functions in FairCom's c-treeACE API. Consult application documentation or application vendor for further login instructions.

It is important to be aware that the file security provided by the c-treeACE Server is a function of access to files through the c-treeACE Server. When files are not controlled by the c-treeACE Server, they may not be secure.

4.1 Users, Files, Groups, and File Permission Masks

This section covers the security concepts needed to understand and make use of the full range of Administrator security controls offered by the c-treeACE Server. These security features are designed to work together. For example, security instructions can be arranged allowing only certain sets of users particular rights with respect to a given file.

See Also

- Security Administrator Utility - sa_admin (page 76)

Users

Whenever an application connects to a c-treeACE Server, it must identify itself to the c-treeACE Server. The identifying code is called the User ID. To gain access to the c-treeACE Server, the User ID seeking access to the c-treeACE Server must be one already authorized as a valid User ID. A password for the User ID may also be required to access the c-treeACE Server.

If one attempts to log on to a c-treeACE Server with an invalid User ID (i.e., one not issued by the Administrator or created by changing an existing User ID), the c-treeACE Server will deny the request and send a message to that effect (i.e., error message 450). An attempt to log on with a valid User ID but an invalid user password will also be denied, with a message stating the reason (i.e., error message 451).

When an application, i.e., a user running a given application, logs on to the c-treeACE Server, a task user is created to identify the session with the User ID. This is relevant when monitoring or disconnecting clients from the c-treeACE Server.

The c-treeACE Server recognizes four kinds of users:

Administrator

The Administrator, or “super user”, is the only user with a pre-set, and unchangeable, User ID (ADMIN). By having the only initial valid User ID, ADMIN is the first user to gain access to the c-treeACE Server. After changing the password for User ID ADMIN from the initial password, ADMIN, to a secure private password, the Administrator uses the ADMIN User ID and the private password to obtain exclusive access to the Administrator utilities needed to carry out the responsibilities discussed in this manual.

Unique User ID

The Administrator can create new User IDs (and passwords) for other users, who then log onto the c-treeACE Server with these names. This includes new members to the ADMIN group with limited Administrator capabilities.

Application-based User ID

Application programs can be designed to supply a given User ID code when attempting to log on the c-treeACE Server, regardless of who the user is. This User ID is treated like a unique User ID, although several users may share a common ID. In other words, the application/user is allowed onto the c-treeACE Server only if the User ID (and the password, if any) supplied to the c-treeACE Server has been authorized.

Guest Users

Users without a unique User ID. An application program can be designed to log onto a c-treeACE Server without requiring the user to supply a User ID and without supplying an application-based User ID. When no User ID is supplied to the c-treeACE Server as an application logs onto the c-treeACE Server, the c-treeACE Server automatically assigns the special User ID “GUEST” to that session.

A V10 Server does not accept guest logons by default. To allow guest logons, add: `GUEST_LOGON YES` (page 169).

User IDs can be up to 31 characters long. Characters can be letters, numbers, or punctuation marks. User IDs are not case sensitive (i.e., upper and lower case characters are treated as the same).

User passwords can be up to 63 characters. (Nine characters for V9 and prior.) Characters can be letters, numbers, or punctuation marks. Passwords are case sensitive (i.e., upper case and lower case characters are treated as different).

Note: Users, including ADMIN, can use the **ctpass** utility (see "User's Control of Security Options" (page 54)) to change their own password. Members of the ADMIN group can use the c-treeACE Server Administrator Utility (page 56), described below, to change the password for a User ID that is not a member of the ADMIN group; only the super ADMIN account (named ADMIN) can change a password for an account that is a member of the ADMIN group.

User ID and Membership in Groups

The Administrator can establish groups of any sort (e.g., a payroll group, a shipping room group, a data entry group) and associate each User ID to as many as 128 of these groups. For example, User ID "B.Smith" is a member of Group ID "Payroll". These connections are ordered, from the "1st" to "Nth" group membership, where N is a maximum of 128.

If the Administrator does not assign a given User ID to a group, the c-treeACE Server automatically assigns that User ID to a special group with the GUEST Group ID. In addition, the special GUEST User ID is automatically assigned to the GUEST group.

A primary (i.e., default) group is always defined for each User ID. This is either the Group ID for the first association or, if no Administrator established associations, the GUEST group. For instance, number 1 on the list of 128 possible connections between the user and groups set up by the Administrator.

These group mechanisms are important in connection with the file permission masks. See "Groups" (page 48) for more information.

User ID and Ownership of Files

Each file created by the c-treeACE Server has an owner. The User ID in effect when a file is created is automatically made the owner of the file, but the Administrator can later change a current file owner to any other valid User ID. The concept of file owner is important because it can be used with the file permission mask. See "Files" (page 47) for more information.

User ID and Logon Limits

The Server Administrator can set several system-wide limits and User ID overrides for those limits. The number of consecutive logon failures, the delay after failure limit is reached, and a minimum time between logons can all be set system-wide with configuration keywords. These settings can be overridden for each User ID using the Server Administration utility, **ctadmn**, which can also set beginning and ending dates for each User ID. These features are detailed below and in *c-treeACE Server Administrator Utility* (page 56).

The Server Administrator can set an optional limit on the number of consecutive failed logons that will cause subsequent logon attempts to fail for a specified time interval. The default logon limit is zero (0), which implies the feature is not active. Logons are blocked for 5 minutes by default after exceeding the limit. A logon during this period returns **LRSM_ERR** (584). Set the logon limit with

`LOGON_FAIL_LIMIT <logon limit>` in the configuration file. The length of time the logons are blocked is set by `LOGON_FAIL_TIME <minutes>` in the configuration file.

The c-treeACE Server can be configured to require user logons within a given period. This ensures all users log on “at-least-once” within the defined time (e.g., at least once a week). If the time expires for a specific user, the server deactivates the user’s profile, preventing access to the server. The Server Administrator, or other ADMIN group user, must reset the user’s account once the time limit has elapsed. To activate this feature, add the following keyword in the server configuration file, *ctsrvr.cfg*, where *<value>* is the period in minutes during which the user must logon:

```
LOGON_MUST_TIME <value>
```

Files

Database files have several security features in addition to the file permission mask, discussed in a separate section:

File Password

Files created by the c-treeACE Server, and others, can be assigned a file password when created. File passwords can be changed later by the Administrator or the file’s owner, and then be required for users to access files. For example, a user could be required to enter a file password before initiating the file operations specified in the file permission mask (see "File Permission Masks" (page 48)).

File passwords can be up to 9 characters long. Characters can be letters, numbers, or punctuation marks. Passwords are case sensitive (i.e., upper case and lower case characters are treated as different).

File Owner

As explained in "Users" (page 45), when a file is created by the c-treeACE Server, the User ID requesting the creation is established as the owner of the file. The Administrator may change the file owner any time to any other currently valid User ID. The owner is used to define one of the ways file permissions can be granted, e.g., the owner typically has permission to write to the file.

File Group

When created, a file is typically associated with the current primary group of the User ID who created the file. The file group is designed for use with the file permission mask. This can be changed later to any other currently valid Group ID for that User ID by the Administrator or owner. For example, the file permission mask may allow “group permission” to read the file, while no others can (see "File Permission Masks" (page 48)).

If instructed by the user’s application when it creates a file, a file’s Group can be any one of the owner’s other Group IDs, instead of the owner’s primary Group.

The current Owner of a file may use the **ctfile** utility, after entering both the current User ID password and the current file password, to change: the file password; the file permission mask (see "File Permission Masks" (page 48)); the file Group; and even the file Owner itself, which would block the user from accessing the file through the original Owner User ID. "User’s Control of Security Options" (page 54) contains a further description of this treatment.

Groups

A Group is an arbitrary category of associated User IDs and files. For example, a business wanting to separate the payroll department and the shipping department could establish a “shipping” Group and a “payroll” Group and associate appropriate User IDs with one or more of these Administrator-defined Groups. By establishing and using groups, the Administrator can offer file-level operation control to selected groups of users. For example, by using Groups along with file permission masks it is possible to enable users in the payroll department to read, but not write, to any file created by anyone else in the payroll department.

Two Kinds of Groups

The c-treeACE Server maintains a guest group, to which User IDs are associated if they are not assigned to any Administrator-defined Group ID. This means every User ID is associated with at least one group (i.e., the GUEST Group or a Group ID).

The Administrator can create any number of Groups each of which has a Group ID, a text description (for display), a memory allocation specification, and a list of User IDs associated with the Group ID. As noted, the Administrator can associate a given User ID with as many as 128 Group IDs. A GUEST User cannot be associated with any Group IDs; instead, the c-treeACE Server automatically assigns a GUEST User to the GUEST Group.

Group IDs can be up to 31 characters long. Characters can be letters, numbers, or punctuation marks. User IDs are not case sensitive (i.e., upper and lower case characters are treated as the same).

File Permission Masks

Once a user has access to a given file, which might need both user and file passwords to reach, there is one additional level of access control available. This is the “file permission mask,” a set of controls over who can do what with a given file. The “what” and the “who” of file permission masks follow.

Operations controlled

User permissions with respect to the following file operations can be controlled with the file permission mask for a given file (i.e., “YES, TYPE X USERS have permission to do this operation” or “NO, TYPE X USERS do not have permission to do this operation”):

- READ the file
- WRITE to the file (i.e., add, update, or delete individual items in the file)
- CHANGE THE DEFINITION(s) of the file, including such characteristics as alternative collating sequences or record schemas (see the *c-treeACE Programmer’s Reference Guide* for details)
- DELETE the entire file
- Any combination of the above

If a file has no permission mask, any user who can access the file can perform all the above operations.

User Controls

Each of these permissions for a given file can be specified for any or all of the following classes of users:

- WORLD access: Allow the specified file operations to any user who can access the file (so users who lack a required User ID and/or file password do not have these file-operation permissions).
- OWNER access: Allow the specified file operations to the current owner of the file. The owner is either the User ID in effect when the file was created or a different User ID who was later assigned as the owner.
- GROUP access: Allow the specified file operations to any User ID currently a member of the same Group as the current File Group.

In summary, a file permission mask permits different degrees of access to a file for the file's owner, users belonging to the file's group, and all other users, including guests.

Using the concepts discussed above, the Administrator can establish a sophisticated and flexible security system with the c-treeACE Server. The mechanism for actually entering information for use by the c-treeACE Server is a separate program utility, called the Administrator's Utility, **ctadmn**.

Informing Users of their Security Options

Users can change the password for their own User ID and they can change security controls for a file if they are the owner of the file. To optimize the use of c-treeACE you may wish to be sure users are aware of these abilities, and how to appropriately apply them.

See "User's Control of Security Options" (page 54) for details.

4.2 Dynamic Advanced Encryption

FairCom offers developers several advanced encryption routines, including AES (Rijndael), Blowfish, Twofish, and DES. Advanced encryption must be enabled at runtime via a sever configuration keyword. The choice of encryption algorithm and cipher strength is a per-file choice by the application developer at file creation time. A master password is then assigned to the server installation which must be provided in some form at server startup.

When advanced encryption is enabled, c-treeACE prompts for a master password at server startup by default. For high availability, options are available to use a local key store file to maintain and verify the master password. The system administrator may encrypt existing files using the **ctcv67** utility.

Developers can also implement the c-treeACE Server SDK to replace this prompt with an application-specific method of retrieving the master password.

Note: Prior to enabling advanced encryption, understand that there is no practical way to recover encrypted data without knowing the master password that was used to encrypt it. This applies to backed up data as well as live data. If a master password is changed, be sure to retain the old master password for any backups that may still be encrypted with the previous master password.

Enabling Advanced Encryption Support

Follow these steps to enable advanced encryption support:

1. When Advanced Encryption is enabled, c-treeACE requires a master password at server startup. Run the **ctcpvf** (page 86) utility to generate a new master password for use when launching the Advanced Encryption enabled Server. This will generate the file *ctsrvr.pvf*.

   Note: Developers can use the c-treeACE SDK to replace this prompt with an application-specific method of retrieving the master password.

2. To enable Advanced Encryption, place the following keyword in the *ctsrvr.cfg* configuration file prior to launching:

   ```
   ADVANCED_ENCRYPTION YES
   ```

Important: Advanced Encryption is disabled by default. Any time you change the advanced encryption setting, you should delete the *FAIRCOM.FCS* file (which contains user and group information) before restarting c-treeACE so user and group information is encrypted for protection, as well. All user and group information must be recreated if the *FAIRCOM.FCS* file is deleted. Alternatively, **ctcv67** can be used with option *E* to encrypt an existing *FAIRCOM.FCS*.

See Also

- Master Password Verification File Utility - ctcpvf (page 86)
- Change Master Password Utility - ctencrypt (page 88)

Encrypting Files Using Advanced Encryption

Client implementation of Advanced Encryption is accomplished through the use of the **SetEncryption()** function on a per file basis. Refer to the *c-treeACE Function Reference Guide* for details on this function. Refer to the *c-treeACE Programmer's Reference Guide* for complete details on implementing advanced encryption.

See Also

- Master Password Verification File Utility - ctcpvf (page 86)
- Change Master Password Utility - ctencrypt (page 88)

Changing the Master Password

You can use the standalone **ctencrypt** utility or the **ctadmn** utility to change the master password. Using the **ctadmn** utility, the **Change Server Settings** menu has an option to **Change advanced encryption master password**. This will Quiesce the server and update the master password for all files or a provided list of files, plus some server-controlled files like *FAIRCOM.FCS*. Using **ctadmn** to change the master password requires the `ALLOW_MASTER_KEY_CHANGE YES` option to be specified in *ctsrvr.cfg* (default: NO).

In V11 and later, a function can be used to change the password that is used to encrypt the file-specific encryption keys in the specified files. The function is supported by the c-treeACE Server and by the standalone c-tree **ctencrypt** utility.

Changing the master password in client/server mode

There are two ways to change the master password in client/server mode:

- Use the **ctadmn** utility
- Use the **SECURITY()** function

Use the ctadmn utility

1. Select option **10. Change Server Settings**.
2. Select option **7. Change advanced encryption master password**.
3. Enter the name of a file on the client system that contains the names of the c-tree data and index files that are to be modified. The file is a text file that contains one filename per line. Any names of transaction logs that are specified in this file are ignored. (The c-treeACE Server automatically locates its active, inactive, and template transaction logs and updates them.)
4. Enter the current advanced encryption master password.
5. Enter the new advanced encryption master password. **ctadmn** prompts twice for the new password to confirm that it was entered correctly.

If the c-treeACE Server successfully changes the master password for all the specified files, **ctadmn** displays the message:

```
Successfully changed the advanced encryption master password
```

If an error occurs, **ctadmn** displays the following message:

```
Error: Failed to change the advanced encryption master password: <error_code>
```

where `<error_code>` is the error code indicating the cause of the failure.

In case of an error, check *CTSTATUS.FCS*, as it might contain more descriptive messages that explain the cause of the error.

Use the SECURITY() function

To change the master password using the **SECURITY()** function:

1. Call the **SECURITY()** function with the *SEC_CHANGE_ADVENC_PASSWD* mode.
2. Specify *filno* of *-1*.
3. Set *bufptr* to point to a buffer that holds the master password change information and set *bufsiz* to the size of the buffer.

   The buffer must conform to the *ctENCMOD* structure definition shown below:

   ```
   typedef struct ctencmod
   { LONG options; LONG numfiles; TEXT varinf[4]; }
   ctENCMOD, *pctENCMOD;
   ```

4. Set *options* to *ctENCMODlowl*
5. Set *numfiles* to the number of files whose names are specified in the *varinf* field (do not include the current and new master passwords in this count even though those values are also specified as the first two strings in the *varinf* field).
6. In the *varinf* field, store the following values as null-terminated strings:
   - the current master password

- the new master password
- the first c-tree file name
- the second c-tree file name
- ...
- the *N*th c-tree file name (where *N* equals *numfiles*)

When using the c-treeACE Server master password change interface, c-treeACE Server attempts to change the master password for the specified files and for all active, inactive, and template transaction logs that it knows about. If any of the files cannot be changed, the entire operation is undone. When the entire operation is successful, the *ctsrvr.pvf* file is also updated using the new master password.

If an error happens on the transaction logs but the c-treeACE Server terminates before it can undo the changes, some files may be left using the new master password but the master password is still set to the old value. In this case, the **ctencrypt** standalone utility (see *Changing the master password using the ctencrypt standalone utility*) can be used to change the master password for those c-tree data, index, or transaction log files that need to be changed.

Error Codes

Two error codes have been added:

Value	Symbolic Constant	Interpretation
932	**BMPW_ERR**	The specified encryption master password is incorrect.
933	**ICOD_ERR**	An encryption operation failed due to an unexpected internal error. See *CTSTATUS.FCS* for details.

See c-tree Error Codes (http://docs.faircom.com/doc/ctreeplus/#28320.htm) for a complete listing of valid c-tree error values.

This also requires that the `ALLOW_MASTER_KEY_CHANGE` , `http://docs.faircom.com/doc/ctserver/#68783.htm` configuration option is enabled, as explained in the *c-treeACE Server Administrator's Guide*.

Changing the master password using the ctencrypt standalone utility

ctencrypt is a standalone utility that can be used to change the master password for the specified c-tree data, index, and transaction log files. Below is the command-line usage for this utility:

```
ctencrypt <options> <command>
```

Supported options:

- *-n <sect>* - Specify node sector size. The default is 64, which corresponds to `PAGE_SIZE` of 8192.

Supported commands (only one at a time may be specified):

- *-chgmpw <filelist>* - Change master password for the files whose names are listed in the file *<filelist>*.

<filelist> is the name of a text file created by the end user that lists the names of the files, one per line, that are to be processed.

ctencrypt requires a password verification file named *ctsrvr.pvf* that was created using the current master password to exist in its working directory. **ctencrypt** prompts the user for the current master password and for the new master password (prompting twice to confirm that the new password was properly entered). Then **ctencrypt** processes the specified files, indicating the status of each file and the total of successful and failed operations.

Unlike the c-treeACE Server master password change operation, **ctencrypt** does not undo any changes in case of an error. The files that it lists as successfully updated will use the new master password even if the utility failed to update other files. Also, if you wish to use the **ctencrypt** utility to modify any transaction logs, their names must be specified in the list file. **ctencrypt** does not attempt to locate any transaction log files on its own (as the c-tree Server operation does).

ctencrypt creates a temporary directory named *temp\ctencrypt.tmp.<process_id>* to store its transaction logs. This directory is normally deleted when **ctencrypt** shuts down.

Below is sample output from ctencrypt:

```
c-treeACE(tm) Version 9.5.35095(Build-101118) c-tree file encryption utility

Copyright (C) 1992 - 2010 FairCom Corporation
ALL RIGHTS RESERVED.

This utility requires a master password in order to start.
Please enter master password:

Enter new master password   :
Confirm new master password :

Changing master password for the specified files...

[  OK   ] SYSLOGDT.FCS
[  OK   ] vcusti
[  OK   ] L0000000.FCT
[  OK   ] L0000002.FCA
[  OK   ] L0000003.FCA
[  OK   ] L0000004.FCA
[  OK   ] L0000005.FCA
[  OK   ] L0000006.FCS
[  OK   ] L0000007.FCS
[  OK   ] L0000008.FCS
[  OK   ] L0000009.FCS
[  OK   ] L0000010.FCT

12 succeeded, 0 failed
Successfully changed master password for all specified files
```

5. User's Control of Security Options

Users, including the super-user ADMIN, can add or change their password. The user who is owner of a file can change file security information for the file. Utilities for implementing user security options are described here.

5.1 The User's Password

The following steps are required for a user to change the password associated with their own User ID:

1. Run the utility program **ctpass** as any other program in the environment.
2. Enter your current User ID.
3. Enter the current password for your User ID, if you have one. (Maximum 63 characters. Maximum nine characters for V9 and prior).
4. Continue by entering the current name of the c-treeACE Server (i.e., the default name or another name, supplied in the c-treeACE Server configuration file).
5. Now change your password by entering the new password.
6. To be sure to enter the new password, you may be asked to enter it twice before it will be accepted. If the same name is not entered both times, try again.

Note: Whenever input is requested, the user may enter a question mark (?) to receive HELP.

After the new password is entered and confirmed, a message saying your User ID password has been successfully updated will be displayed. After being updated successfully, the new password must be used with the User ID to log on to the c-treeACE Server.

Note: All users can change their own passwords. In addition, users who are members of the ADMIN group can change the password of all accounts that are not members of the ADMIN group. Only the super ADMIN account (named ADMIN) can change a password for an account that is a member of the ADMIN group.

5.2 File Security Controls

The owner of a file can change the security information for their file, as follows:

1. Run the utility program **ctfile** the same way as any other program in the environment.
2. Enter current User ID.
3. Enter the current password for the User ID, if one has been assigned.
4. Continue by entering the c-treeACE Server's current name, which is either the default name or another name specified by the server configuration.
5. Now give the name of the file whose security information is to change.
6. If the named file has a file password, the next step is to enter the password.

Note: Whenever input is requested, the user may enter a question mark (?) to receive HELP.

The file owner may change the following security options for their file:

- Change the file's password.
- Change the file's permission mask.
- Change the file's Group.
- Change the file's Owner

Caution! Be careful changing owner security. Once the owner has been changed, then the original owner may no longer use the utility, **ctfile**, to access the file and change security information).

Note: The Administrator can always use the Administration Utility to change, or view, the file security information for any file controlled by c-treeACE.

Default Permissions

c-treeACE defaults to a permission mode of 0660 (read/write access for owner and group; no access for world) for the files it creates.

When using c-treeACE the permission mode assigned to files can be set with the server configuration keyword `FILE_CREATE_MODE` to specify the desired file permission mode.

Example

```
;Set read and write permission for owner
;and no permission for group and world.
FILE_CREATE_MODE   0600
```

Note: On Unix systems, the system's umask setting also affects the permission mode assigned to a file when it is created. If the umask setting is non-zero, the specified permissions are removed from the file's permission mode.

6. Administrator Utilities

Once the c-treeACE Server is installed on the operating system, it is ready to be used. Starting and stopping the c-treeACE Server are basic Administrator responsibilities, therefore this chapter is required reading.

6.1 c-treeACE Server Administrator Utility

The c-treeACE Server Administrator Utility, **ctadmn**, is used by an Administrator to manage general server operations including configuring users, groups and files. It can also be used to monitor the users logged on to the c-treeACE Server and to disconnect any user from the c-c-treeACE Server.

To use this utility, do the following (which may vary between environments, depending on user-interface specifics):

1. Start running the program, **ctadmn**, as any other program in the environment.
2. Enter an ADMIN group User ID. Initially, only ADMIN will exist until you create others.
3. Supply the current password for the User ID given.

   Note: If this is the first time **ctadmn** has been run and the password for the User ID ADMIN is still ADMIN, change it before leaving **ctadmn** to ensure secure access to this program in the future. You can also change your password using **ctpass** (see "Controlling c-treeACE Server Access" (page 44)).
4. You will be prompted for the (optional) file password for the file *FAIRCOM.FCS*. We recommend you do NOT give this file a password since you should already be the only one who can run this utility. To confirm the absence of a password, press the return key.
5. In response to the next prompt, supply the current name for the c-treeACE Server and press enter. If the c-treeACE Server name has not been changed (see "Basic Keywords" (page 161) for details on `SERVER_NAME`), simply press Enter to use the default name.

After finishing these steps, the main menus for the c-treeACE Server Administrator Utility will be displayed, allowing access to the following groups of operations:

1. User Operations
2. Group Definitions
3. File Security
4. Monitor Clients
5. Server Information (*IOPERFORMANCE*)
6. Server Configuration (*SystemConfiguration*)
7. Stop Server
8. Quiesce the Server
9. Monitor Server Activity
10. Change Server Settings

The following sections detail the areas controlled by the Administrator Utility. The steps necessary for each operation may vary slightly in different environments.

Note: While using the Administration Utility, press the question mark key ('?') at any prompt to access Help.

User Operations

User operations available in **ctadmn** are as follows:

- Add New User by:
  - Entering a new User ID.
  - (optional) Enter a long name (i.e., a text string) for use as a User Description (e.g., for screen displays, where the User ID may be too terse).
  - (optional) Enter a User Memory Limit, which is a maximum memory allocation for this particular user that will override maximum memory allocations set at the server-level (for any user) or at the group level (for any particular group).
  - (optional) Enter the User Memory Rule: Absolute, Guideline, or Default. See `USR_MEM_RUL` in "Configuring the c-treeACE Server" (page 153) for details.
  - (optional) Enter a user password.
  - (optional) Assign user membership in from 1 to 128 Groups (i.e., Group IDs).
  - (optional) Enter the first valid date for this User ID.
  - (optional) Enter the last valid date for this User ID.
  - (optional) Enter limit on consecutive logon failures if different from system default. See `LOGON_FAIL_LIMIT` in "Miscellaneous Control" for details.
- Remove an existing User ID.
- List authorized User IDs.
- Change the Password for a given User ID.
- Change which Group(s) a User ID is associated with by adding (up to 128) or removing groups from a list of a User's association with Group IDs.
- Change the User Description, i.e., change the long name identifying the User ID.
- Change User Memory. Change the maximum amount of c-treeACE Server memory a given user can consume.
- Change Extended Logon Validation, including start date, end date, and consecutive logon failures.
- Change User Logon Limit limits users to a specified number of concurrent logons based on their user name. By default, the limit is zero, meaning no limit.

Group Definitions

Group definition operations available in **ctadmn** are as follows:

- Add New Group by:

  - Entering a new Group ID.
  - (optional) Entering a long name (i.e., a text string) for use as a Group Description (e.g., to display, where the Group ID may be too terse).
  - (optional) Entering a Group Memory Allocation, which is a maximum memory allocation for Users who are a member of this particular Group, and will override maximum memory allocations set at the server-level (for any user).
- Remove an Existing Group ID.
- List Groups: List all current Group IDs and descriptions.
- Change Group Membership: Add or remove User IDs from a given Group.
- Change Group Description: Change the long name for the Group ID.
- Change Group Memory: Change the maximum amount of Server memory user in a given group can consume.
- Change Group Logon Limit: Limit users to a specified number of concurrent logons based on their group membership. By default, the limit is zero, meaning no limit.

File Security

File security operations that can be performed on a given file using **ctadmn** are as follows:

- Change the File's Password.
- Change the File's Permission Mask, controlling file operation permissions for three classes of users: World, Group, and File Owner.
- Change the File's Group.
- Change the File's Owner.

Note: Applications can be designed so separate data files and/or index files can be joined into a "superfile," which is a single physical file from the point of view of the operating system. Separate "logical" files within a superfile are called superfile member files. From the point of view of the Administrator, superfiles, member files, and separate data or index files are all treated the same way.

Monitor Clients

The Administrator may want to know which users are currently attached to the c-treeACE Server or may want to force a user to disconnect from the c-treeACE Server. These functions are available:

- List Attached Users.
- Disconnect Users.

Note: Users IDs are associated with Task users (i.e., sessions). It is a task, or session, that is actually terminated. If a User is disconnected using **ctadmn**, the *CTSTATUS.FCS* entry is augmented by the terminated user ID and node name.

Server Information

This prompt provides performance information since the last startup of the c-treeACE Server.

The following is an example of the server performance statistics:

```
============================================================
============  Server performance statistics  ================
============  Values are since last startup  ================
============================================================

        data buffer requests   =          382
        data buffer hits       =          364
        data buffer hit rate   =         95 %

        index buffer requests  =           77
        index buffer hits      =           73
        index buffer hit rate  =         94 %

        # of read operations   =          113
        bytes read             =       226761
        # of write operations  =           20
        bytes written          =        91264

============================================================

Press RETURN to continue...
```

Server Configuration

This prompt provides configuration information since the last startup of the c-treeACE Server.

Note that c-treeACE developers can retrieve much of this information using the **SystemConfiguration**() function.

The following is an example of current server resource values:

```
================================================================
============  Current Server Resource Values   ================
================================================================

        current system memory usage              =  239526744
        highest system memory use                =  241289947
        current system net allocations           =      24238
        c-treeACE files opened by system         =         15
        physical c-treeACE files open            =          6
        c-treeACE file control blocks in use     =         18
        # message in delete node queue           =          0
        # message in checkpoint queue            =          0
        # messages in system monitor queue       =          0
        current # of logons                      =          1
        current # of pending locks (system wide) =          0
        maximum number of logons                 =         32
        limit for the maximum number of logons   =         32
        maximum number of client nodes           =   no limit
        maximum number of connections per node   =   no limit
        remote connections allowed               =        YES
        c-treeACE Server version                 = V9.0.10459
================================================================
Press RETURN to continue...
```

Beginning with c-treeACE V10.3, the server's "mini" version number, build date, and base build date (if any) are now available in the version string.

Stop Server

This prompt allows the Administrator to stop the c-treeACE Server. **ctadmn** will ask for verification that the c-treeACE Server is to be stopped and ask for a shutdown delay in seconds.

Quiesce Server

The Quiesce Server option allows an administrator to immediately block all access from clients to the server while maintaining a consistent server state.

```
Successful Quiesce.

It is now safe to perform a system backup of c-treeACE Server's controlled files.
Press RETURN once the backup is completed to resume the c-treeACE Server.

Press RETURN to continue...
```

While in this state all data and index files are physically closed and can be safely copied, backed up, or moved. Simply press RETURN to bring the server back online to clients.

When you quiesce the server, as long as the connection that quiesced the server remains connected, all other connections are blocked. Only if that connection goes away do we allow the ADMIN user to logon again and undo the quiesce.

ctadmn utility checks for active transactions before quiescing c-treeACE Server

When the administrator selects the **ctadmn** utility's option to quiesce a c-treeACE Server, **ctadmn** checks for active transactions. If any transactions are active, **ctadmn** prompts the user for a maximum time to wait for transactions to complete. That value is passed to the **ctQUIET()** function, which waits up to the specified number of seconds before aborting active transactions and quiescing c-treeACE Server.

Monitor Server Activity

From this menu an administrator can quickly obtain lists of files, including by connection, as well as locks on a particular file. These options are useful in determining active file usage by applications.

```
Monitor Server Activity:

1. List all files open by the c-treeACE Server
2. List all files open by a particular connection
3. List connections that have a particular file open
4. List locks held on a particular file
5. Close a file that is open in KEEPOPEN mode

Enter your choice (1-5), or 'q' to return to previous menu>>
```

Change Server Settings

c-treeACE allows certain options to be enabled/disabled at run time. This **ctadmn** menu displays available options. As the list grows over time, you may find additional entries not specifically listed here.

```
        Change Server Settings:

                1. Configure function monitor
                2. Configure checkpoint monitor
                3. Configure memory monitor
                4. Configure request time monitor
                5. Change dynamic dump sleep time
                6. Enable or disable a status log mask option
                7. Change advanced encryption master password
                8. Change a DIAGNOSTICS option
Enter your choice (1-8), or 'q' to return to previous menu>>
```

An option to set the value of `DYNAMIC_DUMP_DEFER_INTERVAL` (page 279) has been added to this menu.

ctadmn user listing for rtexecute thread running report launched by RTSCRIPT

The Communications section of the user listing shows **rtexecute** for threads launched by RTSCRIPT calls. Below is an example listing:

```
UserID: ---                              NodeName:
Task 15                                  Communications: rtexecute
         Memory: 1512K     Open Files: 8     Logon Time:   --
         Tran Time:   --   Rqst Time:  0:30  InProcess  Rqst# 0   -unknown-
```

This is analogous to how the launched dynamic dump thread, **idyndmp**, is listed. The listing states which thread to kill if one is using **ctadmn** to stop a report. Killing the thread that shows RTSCRIPT as the last function called will not interrupt the report being compiled by **rtexecute**.

6.2 ctstop - Server Stop Utility

Usage

```
ctstop [ -auto ] [<ServerName> <AdminId> <AdminPassword>]
```

This utility shuts down a c-treeACE Server.

Passing the *-auto* switch to **ctstop** without specifying a *ServerName*, *Password*, or *AdminID* shuts down a c-treeACE Server with the defaults shown below:

```
ctstop -AUTO FAIRCOMS ADMIN ADMIN
```

The **ctstop** utility supports passing in *ServerName*, *Password*, and *AdminID* when the *-auto* switch is used.

The optional *delay* value is the number of seconds to wait before shutting down, with a default of no delay.

6.3 ctstat - Statistics Utility

The c-treeACE Statistics Utility, **ctstat**, is a client utility used to display statistics collected by c-treeACE. **ctstat**, provides valuable real time monitoring of critical c-treeACE 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 Connection Handle
-vat	Admin-Transaction Report
-var	Admin-Replication Reader Report
-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
-sqlidx	Shows index scan statistics
-sqlcache	Shows SQL cache information
-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 | reset	Turn function call timing on or off or reset the function-timing statistics
-mf logfile	Log all memory allocations to the specified file
-ma logfile	Log aggregate memory allocations to the specified file
-mr min,max	Log only memory allocations in the range min,max
-ms	Output memory allocation statistics
-mu	Unload module debug symbols
-filelocks datafile	List all locks on a data file
-filelocks file [N]	List all locks on a data file. Displays the Nth key. See **-filelocks Notes** below.
-userlocks user	List all locks held by a user. See **-userlocks Notes** below.

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. In V11 and later, standard output is flushed after the interval determined by *-i* so that output is written to the file immediately. This better handles cases where output is redirected to a file.
-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.

-filelocks Notes

The *-filelocks* option lists all locks on a data file and, optionally, displays the Nth key. The lock offset and the associated keys are not read at the same time. Since we are reading records locked by other users to generate the key, there is no guaranteed relationship between the lock and the displayed key. The following are possible scenarios:

1. The displayed key is from before or after any changes made by the lock holder.
2. The locked offset no longer holds a valid record (it has been deleted, or updated and moved).
3. The locked offset could have been locked/modified/unlocked more than once between the time the lock offset was acquired and the time the record is read, so the offset could hold an entirely different record than what was originally locked.

The *-filelocks file [key]* command supports c-tree's standard wildcard filename matching for the specified file, allowing locks from multiple files to be displayed. The standard wildcards (used by *ctsrvr.cfg* keywords such as MEMORY_FILE and REPLICATE, etc) are:

`*` - Multi-character match

`?` - Single-character match

`^` - Negation (must be first character)

-userlocks Notes

For the *-userlocks* report:

- If *UserID* is a number, it is interpreted as a task ID.
- If *UserID* is a string, it is interpreted as a name, and information on locks held by each task ID with a matching name is returned.

Because the *-userlocks* report may generate a large number of server calls (for each task ID and file), the *-userlocks* report interval may be increased up to 60 seconds, depending on the number of matching users and files involved.

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       files        conne          locks
                      i/o                     ct                          transaction
                                                                                  s

d     %     i     %     r     w     cur/max      cur/m     c     l     %     de    a     t     r     w
%     m     %     m     /     /                  ax        u     %     m     ad    c     /     /     /
h           h           s     s                  r     h                       t     s     t     t

                                    2581/1000                 1           0           3
9     1     9     1     1     1     0            2/128     0     0     0           1     6     0     0
9           9                                                 0
                                    2581/1000                 1           0           3
9     1     9     1     2     0     0            2/128     0     0     0           1     5     0     0
9           9                                                 0
                                    2581/1000                 1           0           2
9     1     9     1     2     0     0            2/128     0     0     0           1     8     0     0
9           9                                                 0
                                    2581/1000                 1           0           2
9     1     9     1     0     0     0            2/128     0     0     0           1     2     0     0
9           9                                                 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

The following command generates a report in text format:

```
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, such as Adds/second, Deletes/second, Updates/second, Reads/second, and totals.

Example

```
# cstat -isam -u ADMIN -p ADMIN -s FAIRCOMS
```

```
add/s   del/s   upd/s  read/s  total/s
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
0,0,0,0,0,0,5,0,0,0,0,0,5,0,0,0,322988,136944,0,0,0,0,mark.idx
0,0,0,0,0,0,5,0,0,0,0,0,5,0,0,0,376483,171571,0,0,0,0,mark.idx M#01
0,0,0,0,0,0,5,0,0,0,0,0,5,0,0,0,0,0,0,0,0,0,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
```

Memory Allocation Example (Windows)

The c-tree Server for Windows can be compiled with an option that causes c-tree's memory suballocator to collect call stacks for each allocation call made through **ctgetmem()**. Each memory allocation is assigned a sequence number. The **ctMEMSTAT()** API function can be used to read the current allocation sequence number and the current number of allocations and to log the call stacks for the allocations to the specified file.

To use this feature, compile the c-tree Server with `#define ctFeatMEMTRACK`. For Windows systems it is ON by default; for non-Windows system, this option is off by default.

The ctstat utility supports the following new memory tracking options:

- *-mf logfile* - Log all memory allocations to the specified file
- *-ma logfile* - Log aggregate memory allocations to the specified file
- *-mr min,max* - Log only memory allocations in the range min,max
- *-ms* - Output memory allocation statistics

Examples:

```
C:\>ctstat -ms -h 10 -s FAIRCOMS
```

Results:

```
  memseq      memalc
    1267         992
    1289         997
    1289         997
    1289         997
```

```
C:\>ctstat -mf memfull.log -i 1 1 -s FAIRCOMS
```

A log of all memory allocations (with each allocation listed separately) is written to the file memfull.log in the c-tree Server's working directory.

```
C:\>ctstat -ma memaggr.log -i 1 1 -s FAIRCOMS
```

A log of all memory allocations (with allocations having the same call stack listed only once each) is written to the file memaggr.log in the c-tree Server's working directory.

```
C:\>ctstat -ma memaggr.log -mr 1900,2000 -i 1 1 -s FAIRCOMS
```

A log of all memory allocations that have sequence numbers between 1900 and 2000 is written to the file memaggr.log in the c-tree Server's working directory.

Note: These options will fail with error 170 if used with a c-tree Server that was built prior to our introduction of this feature and will fail with error 454 if used with a c-tree Server that was built after the introduction of this feature but that was not compiled with #define ctFeatMEMTRACK."

Transaction Statistics Example

Use **ctstat** *-vat* to view the transaction statistics. Sample output is shown below.

- `loglow` is the server's lowest-numbered active log.
- `curlog` is the server's current log.
- `lstent` is the offset in the current log where the last entry was written (which may still be in the in-memory log buffer).
- `lstpnt` is the last byte position written to the log file on disk.
- `lstsuc` is the offset of the last SUCTRAN or CLSTRAN entry in the log.
- `tranno` is the next available transaction number.
- `tfil` is the next available transaction file number.

```
lowlog      curlog      lstent      lstpnt      lstsuc      tranno      tfil
1           4           2702686     2702092     2697872     51806       17298
```

File and User Lock Example

The **ctstat** *-filelocks datafile* and **ctstat** *-userlocks user* reports allow **ctstat** to retrieve lock information by file or by user.

For the *-filelocks* report:

- *-filelocks file [N]* lists all locks on a data file. Displays the *N*th key. Keys are displayed in hexadecimal format following each lock.
- See **-filelocks Notes** below.

For the *-userlocks* report:

- If *UserID* is a number, it is interpreted as a task ID.
- If *UserID* is a string, it is interpreted as a name, and information on locks held by each task ID with a matching name is returned.

Because the *-userlocks* report may generate a large number of server calls (for each task ID and file), the *-userlocks* report interval may be increased up to 60 seconds, depending on the number of matching users and files involved.

-filelocks Notes

The *-filelocks* option lists all locks on a data file and, optionally, displays the *N*th key. The lock offset and the associated keys are not read at the same time. Because the records used to generate the key are locked by other users, there is no guaranteed relationship between the lock and the displayed key. The following are possible scenarios:

1. The displayed key is from before or after any changes made by the lock holder.
2. The locked offset no longer holds a valid record (it has been deleted, or updated and moved).
3. The locked offset could have been locked/modified/unlocked more than once between the time the lock offset was acquired and the time the record is read, so the offset could hold an entirely different record than what was originally locked.

Memory Use and Allocation Call Stacks Example

In V11 and later, it is possible to monitor memory use and allocation call stacks for each suballocator list.

Support has been added for monitoring c-treeACE Server memory use and collecting allocation call stacks for each suballocator list. The ability to monitor c-treeACE Server's memory use has been enhanced in the following ways:

- It now tracks the number and byte count of allocations that do not go through c-treeACE's memory suballocator.
- c-treeACE Server now makes its memory suballocator usage figures available to monitoring tools. The **ctstat** utility's *-ml* option can be used to display current memory allocation figures. Example:

```
# ctstat -ml -t -i 2 -h 1 -s FAIRCOMS

Thu May 01 14:31:22 2014
  name      allocated         in use       pct
KLNTYP              0              0      0.00%
PI1TYP          32768           1704      0.01%
PI2TYP          32768           6800      0.01%
PI4TYP        1376256        1224360      0.31%
PI8TYP        1245184        1155592      0.28%
PIwTYP          32768          19008      0.01%
PIxTYP          32768          16120      0.01%
PIyTYP          49152          25800      0.01%
```

```
PIzTYP          32768              0      0.01%
PIaTYP          82560          69768      0.02%
PIbTYP         296064         270600      0.07%
SHDTYP              0              0      0.00%
BATTYP              0              0      0.00%
ILKTYP          16384              0      0.00%
FNMTYP              0              0      0.00%
COMTYP          16384              0      0.00%
ABTTYP              0              0      0.00%
LOKTYP          16384              0      0.00%
DCMTYP              0              0      0.00%
IXCTYP       32129024       29018360      7.24%
DTCTYP        6782976        6547464      1.53%
CTCTYP          81920          70000      0.02%
IXBTYP      180854968      180854968     40.74%
DTBTYP      203169792      203169792     45.76%
MBATYP       17690815       17690815      3.98%
TOTAL:      443971703      440141151
```

Note: If memory allocation call stack is enabled for a suballocator list, the name of the suballocator list is followed by an asterisk in this output (for example, as MBATYP*).

Just as the **ctstat** utility does, a c-treeACE Server client can read the memory use figures by calling the **ctSNAPSHOT()** API function with the new mode *ctPSSmemAlloc*. This mode returns the memory use figures in a new structure named ctGMMS. See *ctstat.c* for an example of this **ctSNAPSHOT()** call.

- On Windows and Linux systems, c-treeACE Server supports collection of call stacks for memory allocations. This support, which existed prior to this revision, is enabled using the configuration option `DIAGNOSTICS MEMTRACK`. Now c-treeACE Server allows allocation call stack collection to be dynamically enabled or disabled for specific memory suballocator lists, provided that `DIAGNOSTICS MEMTRACK` was specified in the configuration file. The **ctMEMSTAT()** API function is used to change these settings, and the **ctstat** utility's *-mt* option provides a convenient way to use this function.

Only a member of the ADMIN group is allowed to change memory allocation settings.

Examples:

1. Enable memory allocation call stack collection for all suballocator lists:
   ```
   ctstat -mt +ALL -u ADMIN -p ADMIN -s FAIRCOMS
   ```
2. Enable memory allocation call stack collection for only the MBATYP and LOKTYP suballocator lists:
   ```
   ctstat -mt +MBATYP,+LOKTYP -u ADMIN -p ADMIN -s FAIRCOMS
   ```
3. Disable memory allocation call stack collection for all suballocator lists:
   ```
   ctstat -mt -ALL -u ADMIN -p ADMIN -s FAIRCOMS
   ```

As before, the current memory allocations can be logged to a file using the **ctMEMSTAT()** function, as used by the **ctstat** utility's *-ma* option:

```
ctstat -ma mem.log -i 1 1 -u ADMIN -p ADMIN -s FAIRCOMS
```

Compatibility Notes:

1. The use of the existing and new memory monitoring options is now restricted to members of the ADMIN group.

2. When memory allocation call stack tracking is supported on a per-suballocator list basic, the `DIAGNOSTICS MEMTRACK` option must still be specified in *ctsrvr.cfg* to support collecting memory allocation call stacks, but collection of allocation call stacks is initially disabled for all suballocator lists. An administrator must use the **ctstat** utility's *-mt* option as shown above to enable collection of memory allocation call stacks for the desired suballocator lists.

   If troubleshooting unexpected memory growth, first use **ctstat** *-ml* to monitor memory use by suballocator list. Then enable memory allocation call stack collection just for the lists that show the unexpected growth. This approach can reduce the overhead of the memory allocation call stack collection and can simplify analysis of the unexpected memory growth.

6.4 sa_admin - Command-line security administration utility

The command-line version of the system administrator program, **sa_admin**, can be used to perform many user operations directly from shell scripts.

```
sa_admin [-a<adminuserid>] [-p<adminpassword>]  [-f<filepassword>] [-s<servername>] <option>
```

option is one of the following:

Options Users

- *-oua* Add a user account
- *-oud* Change user account description
- *-oue* Change user account extended settings
- *-oug* Add a user to a group
- *-oul* List user accounts
- *-oum* Change user account memory limit
- *-oup* Change user account password
- *-our* Delete a user account
- *-ous* Show user account information
- *-oux* Remove a user from a group

Options Group

- *-oga* Add a group
- *-ogd* Change group description
- *-ogl* List groups
- *-ogm* Change group memory limit
- *-ogr* Delete a group
- *-ogs* Show group information

Options File

- *-ofg* Change file group
- *-ofl* List files matching filename
- *-ofo* Change file owner
- *-ofp* Change file password
- *-ofs* Change file permissions

Wildcard specifiers with sa_admin

sa_admin, *-ofp*, *-ofs*, *-ofg*, and *-ofo* options support specifying filenames with wildcard characters. When one of these options specifies a filename that includes ? or * characters, the utility retrieves a list of files matching the filename wildcard specifier and executes the specified command for each file.

Retrieve a List of Filenames from the server with sa_admin

-ofl (list files) is used to list the files on the c-treeACE Server system matching the specified filename including wildcard characters.

sa_admin Support for Encrypted Password Files

The **sa_admin** utility supports the use of an encrypted password file. Encrypted password files are created with the **ctcmdset** utility and keep user IDs and passwords from plain view within script files. An encrypted password file name is specified using the command-line option:

```
-1 <filename>
```

ADMINISTRATOR OPTIONS

- *-a* System administrator User ID.
- *-p* System administrator password.
- *-f* Optional server system file password.
- *-s* Optional server name.

Note: There is no space between the switch and its parameter.

USER OPTIONS

The following options, all beginning with *-ou*, allow changes to user information. Additional group and file options are described below.

Note: To use any optional entry, you must use all the previous entries even if they would otherwise be optional. For example, to add a user with the *-oua* option and specify a group, you must also enter the *userid*, *desc*, and *password*.

Option User Add

```
-oua <userid> [-d <desc>] [-w <password>] [-g <group>] [-m <memory>[<rule>]]
              [-b <begdat>] [-e <enddat>] [-l <loglimit>] [-r <rsmlogon>] [-t <mstlogon>]
```

- *userid*: User id (required)
- *-d desc*: Optional user description
- *-w password*: Optional user password
- *-g group*: Optional user group
- *-m memory*: Optional user memory limit.
  - *rule*: Optional user memory rule. Used only with memory. The optional *<rule>* is *A* for absolute, *D* for default, or *G* for guideline (example -m 10485760a specifies an absolute memory limit of 10 MB). NULL for Default.

- *-b begdat*: Optional starting validity date. Specify as mm/dd/yyyy. NULL for Default.
- *-e enddat*: Optional ending validity date. Specify as mm/dd/yyyy. NULL for Default.
- *-l loglimit*: Optional maximum invalid logon attempts. NULL for Default.
- *-r rsmlogon* is the logon block period in minutes. Specifying a value of “block” (e.g., *-r block*) blocks the account indefinitely (until it is unblocked by an administrator, and specifying a value of “unblock” (e.g., *-r unblock*) unblocks the account immediately.
- *-t mstlogon* is the interval in minutes during the user must logon at least once, otherwise the account is blocked.

Option User Remove

```
-our userid
```

- *userid*: User id (required)

Option User List

```
-oul
```

Option User Change Password

```
-oup userid password
```

- *userid*: User id (required)
- *password*: New password (required)

Option User Add user to Group

```
-oug userid group
```

- *userid*: User id (required)
- *group*: Group name (required)

Option User (group) Extract - Remove a user from a group.

```
-oux userid group
```

- *userid*: User id (required)
- *group*: Group name (required)

Option User Change Description

```
-oud userid desc
```

- *userid*: User id (required)
- *desc*: New user description

Option User Memory

```
-oum userid memory rule
```

- *userid*: User id (required)
- *memory*: New memory limit. This can be a number of bytes or ‘D’ for default or left NULL for no limit
- *rule*: Optional user memory rule. Used only with memory. This may be ‘A’ for Absolute, ‘G’ for Guideline, ‘D’ for Default, or NULL for Default

Option User Change Extended Settings

```
-oue <userid> [-b <begdat>] [-e <enddat>] [-l <loglimit>] [-r <rsmlogon>] [-t <mstlogon>]
```

- *userid*: User id (required)
- *-b begdat*: Optional starting validity date. Specify as mm/dd/yyyy. NULL for Default
- *-e enddat*: Optional ending validity date. Specify as mm/dd/yyyy. NULL for Default
- *-l loglimit*: Optional maximum invalid logon attempts. 0 for Default. -1 to disable invalid logon check.
- *-t mstlogon*: Optional must logon period, e.g., how often the user must log on to remain active. The interval in minutes during the user must logon at least once, otherwise the account is blocked. Specify as number of minutes. NULL for Default. -1 to disable must logon period.
- *-r rsmlogon*: Optional logon timeout remaining. If a user has been denied access to the c-treeACE Server due to excessive invalid logon attempts, you can adjust the remaining user lockout time here. Specify as number of minutes. NULL to leave unchanged. Specifying a value of “block” (e.g., *-r block*) blocks the account indefinitely (until it is unblocked by an administrator), and specifying a value of “unblock” (e.g., *-r unblock*) unblocks the account immediately.

Option User Show

```
-ous userid
```

- *userid*: User id (required)

GROUP OPTIONS

The following options, all beginning with *-og*, allow changes to group information. Additional user and file options are described elsewhere.

> **Note:** To use any optional entry, you must use all the previous entries. For example, to specify a rule when adding a group with the *-oga* option, you must also enter the *desc* and *memory* options for the group.

Option Group Add

```
-oga <groupid> [-d <desc>] [-m <memory>][<rule>]]
```

- *groupid*: Group id (required)
- *-d desc*: Optional group description
- *memory* is the memory limit and the optional *<rule>* is *A* for absolute, *D* for default, or *G* for guideline (example -m 10485760a specifies an absolute memory limit of 10 MB).

Option Group Remove

```
-ogr groupid
```

- *groupid*: Group id (required)

Option Groups List

```
-ogl
```

Option Group Change Description

```
-ogd groupid desc
```

- *groupid*: Group id (required)

- *desc*: New group description

Option Group Memory

```
-ogm groupid [-m <memory>[<rule>]]
```

- *groupid*: Group id (required)
- *-m memory*: New memory limit. *memory* is the memory limit
  - *<rule>* (optional) is *A* for absolute, *D* for default, or *G* for guideline (example -m 10485760a specifies an absolute memory limit of 10 MB).

Option Group Show

```
-ogs groupid
```

- *groupid*: Group id (required)

FILE OPTIONS

The following options, all beginning with *-of*, allow changes to file information. Additional user and group options are described elsewhere.

Option File Password

```
-ofp filename password
```

- *filename*: File name (required)
- *password*: File password (required)

Option File Security (permissions)

```
-ofs <filename> <permission> ...
-ofs +|-<permission> ...
```

- *filename*: File name (required)
- *permission*: File permission mask.
  To set a permission, set the byte at the corresponding offset to a value of '+'.
  To reset a specified permission, set the corresponding byte to '-'.
  For example, the string "+++++-----+++++" sets all OWNER and WORLD permissions, and clears all GROUP permissions.
  This field is interpreted as a 15-byte permission mask containing owner, group, and world permissions:

```
(offset)
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
----OWNER----  ----GROUP----  -----WORLD----
r  w  f  d  p  r  w  f  d  p   r  w  f  d  p

r = Read  w = Write  f = define  d = Delete  p = noPass
```

- *permission* can also be one of the following:

```
ownerall, ownerread, ownerwrite, ownerdefine, ownerdelete, ownernopass,
groupall, groupread, groupwrite, groupdefine, groupdelete, groupnopass,
worldall, worldread, worldwrite, worlddefine, worlddelete, worldnopass
```

  Options are evaluated left to right. For example, specifying *-groupwrite +groupwrite* has the effect of adding the groupwrite permission, and specifying *+worldall -worldread* turns on all world permissions except read permission.

Option File Group

```
-ofg filename groupid
```

- *filename*: File name (required)
- *groupid*: File group id (required)

Option File Owner

```
-ofo filename owner
```

- *filename*: File name (required)
- *owner*: File owner (required)

Examples of -ofs usage:

`-ofs <filename> <permmask>` is the same as current usage:

```
-ofs test.dat ++++++++++-----
```

`-ofs <filename> <permission> ...` sets the file permissions to the specified permissions. The following command sets all owner and group permissions and resets all world permissions:

```
-ofs test.dat ownerall groupall
```

`-ofs <filename> +|- <permission> ...` adds/removes specified permissions to/from current file permissions. The following command adds the worldread permission to the current file permissions and removes the groupwrite permission from the current file permissions:

```
-ofs test.dat +worldread -groupwrite
```

6.5 ctpass - Password Utility

c-treeACE Server utility to allow users to change their password.

The following steps are required for a user to change the password associated with their own User ID:

1. Run the utility program **ctpass** as any other program in the environment.
2. Enter your current User ID.
3. Enter the current password for your User ID, if you have one. (Maximum 63 characters. Maximum nine characters for V9 and prior).
4. Continue by entering the current name of the c-treeACE Server (i.e., the default name or another name, supplied in the c-treeACE Server configuration file).
5. Now change your password by entering the new password.
6. To be sure to enter the new password, you may be asked to enter it twice before it will be accepted. If the same name is not entered both times, try again.

Note: Whenever input is requested, the user may enter a question mark (?) to receive HELP.

After the new password is entered and confirmed, a message saying your User ID password has been successfully updated will be displayed. After being updated successfully, the new password must be used with the User ID to log on to the c-treeACE Server.

Note: All users can change their own passwords. In addition, users who are members of the ADMIN group can change the password of all accounts that are not members of the ADMIN

group. Only the super ADMIN account (named ADMIN) can change a password for an account that is a member of the ADMIN group.

6.6 ctquiet - Quiesce c-treeACE Utility

The **ctquiet** utility allows an administrator to quiet the server from a script. An interactive option is available in the **ctadmn** utility.

Usage

```
ctquiet [-s server][-f][-u][-w command] {-p password|-a authfile}
```

- *-s* Server (default: FAIRCOMS@localhost)
- *-f* Full consistency - files are flushed to disk from cache (default: crash consistency - no flush). See below.
- *-u* Unquiet server
- *-w* Execute command on successful quiet. Waits for SIGINT to unquiet server
- *-a* Authentication file name
- *-p* Admin Password

The *-a authfile* option takes as argument an encrypted settings file (using **ctcmdset.exe**) with the plaintext form given as:

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

Note: This utility provides a *-f* option, which enables full consistency (also known as a "clean quiesce") in which files are flushed to disk from cache. When this option is used, the system does not require any files to be rebuilt.

The default is the *-f* option is *off*, which results in "crash consistency" (also known as a "dirty quiesce"). The default means that transaction logs are required and, if present, the transaction logs are used to get back into a clean state once recovery completes.

If files are NOT under transaction control, the *-f* option is strongly recommended, otherwise you will have to do a rebuild to get the files back to a clean state.

A quiesced state allows a physical copy of files to be taken that can then be restored at a later time. For systems that provide hardware-based snapshot features, this allows extremely fast system backups to take place while maintaining full data integrity.

Notes

- When you quiesce the server, as long as the connection that quiesced the server remains connected, all other connections are blocked. Only if that connection goes away do we allow the ADMIN user to logon again and undo the quiesce.
- After the server is quiet and the **ctquiet** utility disconnects, one ADMIN connection is allowed to reconnect. There is no prevention of a separate process connecting as ADMIN while the server is in a quiet state and precluding the unquiet call.
- There is a subtle distinction between a "quiet" state, and a file blocked with the **ctFILBLK()** call. While in the quiet state, files are not physically closed and cannot be moved or replaced

while in this mode. Compare this to a "blocked" state, where the file can be replaced, as the OS file handle has been released.

- When the optional *-w COMMAND* switch is used, the behavior of **ctquiet** is modified to perform as follows:
  a. After successfully quieting the server, it makes a system call to execute **COMMAND**.
  b. It remains connected and waits for **SIGINT** to unquiet the server. If **ctquiet** is killed before receiving **SIGINT**, the server will remain in a quiet state until a new connection unquiets the server.
- When connecting to a quieted server with the intent to remove the quiet state when the original caller of **ctquiet** has disconnected you must now set the USERPRF_ADMSPCL bit.

6.7 ctfilbkif - File Block Utility

```
ctfilbkif
```

ctfilblkif will block, or unblock a specified c-treeACE file. The default behavior is to block access to the specified file. Pass the *-u* option to unblock a file.

Usage

```
ctfilblkif [-s server][-f filename][-u] {-p password|-a authfile}

        -s: Server (default: FAIRCOMS@localhost)
        -f: file name
        -u: unblock file
        -a: Authentication file name
        -p: Admin Password
```

See also

- ctquiet - Quiesce c-treeACE Utility (page 82)

6.8 ctdump - Schedule a Dynamic Dump

```
ctdump [ adminuser [ adminpass [ dumpscript [ servername ] ]]]
```

Used to back up data files controlled by the c-treeACE Server.

See also

- *Dynamic Dump* (page 98)
- *ctdump - Dynamic Dump Utility* (page 99)

6.9 ctrdmp - Dynamic Dump Recovery or System Rollback

```
ctrdmp [ dumpscript ]
```

Used to restore dumps created with **ctdump**.

A successful **ctrdmp** completion always writes the following message to *CTSTATUS.FCS*:

```
DR: Successful Dump Restore Termination
```

A failed **ctrdmp** writes the following message to *CTSTATUS.FCS* when **ctrdmp** terminates normally:

```
DR: Dump Restore Error Termination...: <cterr>
```

where *<cterr>* is the error code.

If for some reason **ctrdmp** terminates prematurely (for example, a fatal error causes **ctrdmp** to terminate abnormally), the “Dump Restore Error Termination...” message might not be written to *CTSTATUS.FCS*. In that case, **ctrdmp** might have written error messages to standard output or to *CTSTATUS.FCS* before terminating that helps explain the reason for **ctrdmp** terminating prematurely.

Note: A 32-bit **ctrdmp** could fail with **error 75** if run on transaction logs created by a 64-bit c-treeACE Server, which might support more than 2048 connections.

In V11 and later, the **ctfdmp**, **ctldmp**, and **ctrdmp** utilities display the c-treeACE version used to compile them when they are run.

Environment Variable for Advanced Encryption Password

If this utility has advanced encryption enabled, it can read an encrypted password file instead of being prompted to enter the master password. To enable this, set the environment variable `CTREE_MASTER_KEY_FILE` to the name of the encrypted master password file.

See also

- *Maintaining Database Integrity* (*http://docs.faircom.com/doc/ctserver/8373.htm*) in the *c-tree Server Administrator's Guide*

Rollback to New Restore Points with ctrdmp

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

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

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

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

Now the script can be composed as follows:

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

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

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

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

See also

- *ctrdmp - Dynamic Dump Recovery or System Rollback* (page 83)
- *ctfdmp - Forward Dump Utility* (page 85)

6.10 ctfdmp - Forward Dump Utility

`ctfdmp`

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

ctfdmp takes an `!RP <name>` argument to set the point in time to stop the forward roll. This argument is also used by the **ctrdmp** script option, as described in *Rollback to New Restore Points with ctrdmp* (page 84). To incrementally roll forward from there, rename the previous *RSTPNT_CHKPNT*.FCS* to *S0000001.FCS* (the start point for the **ctfdmp**), and supply a new `!RP` and transaction logs.

> **Note**: The `!` prefix needs to be escaped when using the Unix Bash shell.

In V11 and later, the **ctfdmp**, **ctldmp**, and **ctrdmp** utilities display the c-treeACE version used to compile them when they are run.

Environment Variable for Advanced Encryption Password

If this utility has advanced encryption enabled, it can read an encrypted password file instead of being prompted to enter the master password. To enable this, set the environment variable `CTREE_MASTER_KEY_FILE` to the name of the encrypted master password file.

See also

- *Maintaining Database Integrity* (*http://docs.faircom.com/doc/ctserver/8373.htm*) in the *c-tree Server Administrator's Guide*
- *ctrdmp - Dynamic Dump Recovery or System Rollback* (page 83)
- *Rollback to New Restore Points with ctrdmp* (page 84)

6.11 ctcpvf - Master Password Verification File Utility

c-treeACE advanced encryption (AES, Blowfish, Twofish, 3DES) requires a master password to protect encrypted file access. Before starting c-treeACE for the first time with Advanced Encryption enabled, the Administrator must use the **ctcpvf** utility to create the master password verification file. Each time c-treeACE starts, it prompts for the master password to allow it to open encrypted files.

ctcpvf creates the master password verification file. It accepts optional parameters: filename (the file name to create) and password (the master password). If the parameters are not given, **ctcpvf** will prompt for the required information.

Usage

```
ctcpvf  [-c <cipher>] [-f <filename>] [-k <key>] [-s <store>]
```

Where:

- *-c <cipher>* - Use encryption cipher *<cipher>*. Supported ciphers: aes256 and aes128. Default is aes256.
- *-f <filename>* - Create password verification file *<filename>*. Default is *ctsrvr.pvf*.
- *-k <key>* - Use *<key>* as the master key.
- *-s [<store>]* - Store key in encrypted file *<store>*. Default is *ctsrvr.fkf*.
- *-syslevel* - (V11 and later) Create encrypted store file with system-level encryption: all user accounts on the system can decrypt it.

Note: If you don't use the *-syslevel* switch, you must run the c-treeACE Server under the same user account that was used to run the **ctcpvf** utility that created the master key store file. Using the *-syslevel* switch creates the master key store file so that it can be opened by any user account on that machine, which allows you to run the c-treeACE Server under any user account on the system. (See *Advanced encryption master key store encrypted at system level on Windows* (page 87).)

Note: c-treeACE looks for the file *ctsrvr.pvf* in the server binary area, so this file name should be specified.

Key Store Option

By default, this master key must be presented to c-treeACE on startup as prompted. However, this prompted interaction is not always possible. Consider the case of a failover strategy for business continuity, or the case where no single person should ever know the complete key as keys are built from random secure key generators. c-treeACE supports a key store file to provide this key value at startup.

The **ctcpvf** utility *-s* option is used to select the master key length, and to write the master key to an encrypted keystore file *<store>*.

The c-treeACE configuration option `MASTER_KEY_FILE` specifies the key store file, *<store>*, from which c-treeACE reads the master encryption key. On Linux and Unix systems, the master key is stored AES encrypted in a file on disk, with permissions set such that only the user that created the file can read it (permissions are set to 400). For complete security, it is important to use file system access safeguards to fully protect this key store file.

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

Advanced encryption master key store encrypted at system level on Windows

In V11 and later, support has been added for creating an advanced encryption master key store encrypted at the system level on Windows. Prior to this revision, the encrypted master key store file created by the **ctcpvf** utility on Windows could only be decrypted by the user account that created the file. This made it difficult to set up a Windows service that is using the LocalSystem account to be able to read the encrypted master key store file. (The **ctcpvf** utility had to be run as LocalSystem when creating the master key store.)

An option has been added to the **ctcpvf** utility to create the encrypted store using system-level encryption, meaning that any user account on the system can decrypt the file. Use the **ctcpvf** utility's *-syslevel* option to use this feature. Example:

```
ctcpvf -k mymasterkey -s ctsrvr.fkf -syslevel
```

This option has been added to the **ctadmn** utility's "Change advanced encryption master password" option. Example:

```
Enter the name of the filename list file >> files.txt

Enter the current advanced encryption master password >> ****************

Enter the new advanced encryption master password >> ******************

Please confirm the new master password by entering it again:

Enter the new advanced encryption master password >> ******************

Enter the encryption level [U]ser or [S]ystem for the encrypted store >> u


Changing master password for the specified files...


Successfully changed the advanced encryption master password.
```

See *ctadmn.c* for an example showing how to call the **SECURITY()** function with mode of SEC_CHANGE_ADVENC_PASSWD to change the master key. If you want to create the master key encrypted store using the system-level encryption option, OR in the ctENCMODsysl bit to the options field of the ctENCMOD structure whose address you pass to **SECURITY()**.

Note: This support was added on the Windows platform only.

6.12 ctencrypt - Utility to Change Master Password

The c-treeACE advanced encryption feature uses a master password to encrypt the file-specific advanced encryption key in c-tree data, index, and transaction log files that are encrypted using advanced encryption. **ctencrypt** is a standalone utility that can be used to change this master password for specified c-tree data, index, and transaction log files.

Usage

```
ctencrypt <options> <command>
```

Available Options:

- *-n <sect>* - Node sector size. The default is 64, which corresponds to PAGE_SIZE of 8192.

Available Commands (only one at a time may be specified):

- *-chgmpw <filelist>* - Change master password for the files whose names are listed in the file *<filelist>*. *<filelist>* is the name of a text file created by the end user that lists the names of the files (data and index), one per line, that are to be processed.

ctencrypt requires a password verification file named *ctsrvr.pvf* that was created using the current master password to exist in its working directory. **ctencrypt** prompts the user for the current master password and for the new master password (prompting twice in order to confirm that the new password was properly entered).

Note: ctencrypt does not change the master password file, *ctsrvr.pvf*. The **ctcpvf** utility will need to create a new file for server startup in coordination with the new password used to re-encrypt the encryption key for the files. Failure to do so will result in **DCOD_ERR** errors (606, failure to decode file) when opening files.

ctencrypt processes the specified files, indicating the status of each file and the total of successful and failed operations. Note that the c-treeACE Server must be shut down while these file modifications take place.

ctencrypt creates a temporary directory named *temp\ctencrypt.tmp.<process_id>* to store its transaction logs. This directory is normally deleted when **ctencrypt** shuts down.

Important: ctencrypt does not undo any changes in case of error. The files that it lists as successfully updated will use the new master password even if the utility failed to update other files.

Example File List

A semicolon can be specified at the start of a line to indicate a comment which is ignored.

```
; c-treeACE Advanced Encryption Conversion Listing File
; ------------------------------------------------
; Created Wed Dec 01 01:38:00 2010

; transaction log files
L0000000.FCT
L0000002.FCA
```

```
L0000003.FCA
L0000004.FCA
L0000005.FCA
L0000006.FCS
L0000007.FCS
L0000008.FCS
L0000009.FCS
L0000010.FCT

; data files
  mydatafile.dat
    C:\My Documents\test.dat
vcusti
```

Note: All physical encrypted files, data and index files, must be specified in order to be modified. No attempt is made to determine associated files.

If the server was cleanly shutdown in such a manner that its transaction logs are no longer necessary, then they will not need to be included as part of this password change. If you wish to use the **ctencrypt** utility to modify any existing encrypted transaction logs (for example, archive logs for replication), their names must be specified in the list file. **ctencrypt** does not attempt to locate any transaction log files on its own.

Example Output

```
c-tree file encryption utility

This utility requires a master password in order to start.
Please enter master password:

Enter new master password   :
Confirm new master password :

Changing master password for the specified files...

[  OK   ] SYSLOGDT.FCS
[  OK   ] vcusti
[  OK   ] L0000000.FCT
[  OK   ] L0000002.FCA
[  OK   ] L0000003.FCA
[  OK   ] L0000004.FCA
[  OK   ] L0000005.FCA
[  OK   ] L0000006.FCS
[  OK   ] L0000007.FCS
[  OK   ] L0000008.FCS
[  OK   ] L0000009.FCS
[  OK   ] L0000010.FCT

12 succeeded, 0 failed

Successfully changed master password for all specified files
```

Error Returns

Two new error codes have been added related new password management features:

- **BMPW_ERR** (932) - The specified encryption master password is incorrect.
- **ICOD_ERR** (933) - An encryption operation failed due to an unexpected internal error. See *CTSTATUS.FCS* for details.

6.13 ctvfyidx - Index Verify Utility

The **ctvfyidx** utility uses the **ctVERIFYidx()** function to check the integrity of an index file. The client version of the **ctvfyidx** utility supports the command-line options listed below.

Usage

```
ctvfyidx [<option> ...] [-u <userid>] [-p <password>] [-s <servername] <file name> [<member #>]
```

where *<option>* is one or more of the following:

- *-excl* - Open the file in exclusive mode
- *-delstk* - Check delete stack (on by default)
- *-link* - Check links (on by default)
- *-leaf* - Check leaf nodes (on by default)
- *-chkkey* - Check keys in leaf nodes

The optional parameter *-page size* equals sector size * 128 (third parameter in **InitCtree()**). If page size is not entered, a default value of 16 will be used. *filename* specifies the index file targeted for analysis. The *member #* refers to the index member number. A physical index file can contain one or more indices. Each index has a member number (0, 1, 2, 3, etc.). For example, the sample index file *custordr.idx* provided with the FairCom ODBC Driver contains a total of two indices. Depending on whether you specify 0 or 1 you will be looking at either the order number index or the customer number index. rflg represents an optional recovery flag switch and is only applicable when compiled with *TRANPROC*. Any character will enable *rflg*, which will result in c-tree skipping automatic recovery.

In V10.3.1 and later, the **ctvfyidx** utility defaults to ctREADFIL. It uses ctEXCLUSIVE when the *-excl* option is specified. A ctREADFIL open will fail with error **12** and sysiocod -8 if any connection has the file open for write access.

In versions prior to V10.3.1, **ctvfyidx** will only work with the *-excl* option when connecting to newer servers. Without the *-excl* option, older versions of **ctvfyidx** will fail with **LERR_ERR**.

Example

Below is an example of launching **ctvfyidx** along with output showing the results of the index verification.

```
# ctvfyidx -2048 custmast.idx 0

Index page scan finds entries=4  header=4
Index nodes per level of tree structure - [0: 1]
Internal Index Verify: SUCCESSFUL
```

6.14 ctvfyfil - File Verify Utility

The **ctvfyfil** utility calls the **ctVerifyFile()** function. The utility can be run in standalone and in client/server mode.

Usage

```
ctvfyfil [<option> ...] <file name>
```

where *<option>* is one or more of the following:

- *-chkdat* - Read data file only (default)
- *-chkdatkeys* - Read data file and check keys in index
- *-chkidx* - Read index file
- *-chkidxrec* - Read index file and check records in data file
- *-excl* - Open the file in exclusive mode
- *-int* - Interactive mode: stop on each error
- *-index=<N>* - Check only the *N*th index (N=1,2,...)

Standalone Usage

An additional option is available for standalone use:

- *-<page size>* - Use the specified page size (standalone only)

The example below shows the utility called standalone on a file called *mark.dat* with the page size set to 8192:

```
ctvfyfil -8192 mark.dat
```

7. Backups and Data Integrity

The purpose of any data backup is to protect data integrity. Periodically backing up application data allows a Server Administrator to recover from problems or to roll back a database to a prior point in time.

There are several ways to back up data controlled by a c-treeACE Server:

- Using a standard backup utility while the server is shut down.
- Using the dynamic dump capability while the server is operational.
- Using VSS backup integration on Windows.
- Using ctQuiet/Quiesce for external backups while the server is running.

When possible, FairCom recommends shutting down the server periodically to allow a full backup. This has the advantage of simplicity, since all files can be backed up and restored without using the transaction logs to ensure the data and index files are synchronized. This is especially helpful for applications that do not use transaction control to maintain database integrity. The Administrator can simply restore the files and continue operation.

WARNING: Files under c-treeACE control should never be copied or backed up using third-party software while c-treeACE is operational.

7.1 c-treeACE Server Files

The c-treeACE Server creates special system files to maintain various kinds of information required to recover from problems. The following list details exactly what files are created, along with all required information needed by the System Administrator responsible for working with them. As the Administrator, be sure these files are backed up when appropriate and used for recovery when necessary.

Note: To be compatible with all operating systems, the names for all these files are upper case characters.

c-treeACE Server Status Log

When it starts up, and while running, the c-treeACE Server keeps track of critical information concerning the status of the c-treeACE Server, e.g., when it started; whether any error conditions have been detected; and whether it shuts down properly. This information is saved in chronological order in a text file, the c-treeACE Server Status Log, *CTSTATUS.FCS*. To control the size of *CTSTATUS.FCS*, or to maintain inactive logs as *T*.FCS* files, use the `CTSTATUS_SIZE` keyword. See the keyword description for more detail.

Administrative Information Tables

The c-treeACE Server creates and uses the file *FAIRCOM.FCS* to record administrative information concerning users and user groups. This file can be encrypted with the `ADMIN_ENCRYPT` keyword. See "Configuring the c-treeACE Server" (page 153) for details.

The c-treeACE Server creates the following files for managing transaction processing:

`I0000001.FCS`

Transaction Management Files

I0000002.FCS is an empty file generated at startup by any c-tree database engine with transaction support enabled. This file marks ownership of the process directory to avoid colliding with other c-treeACE processes that may generate their own independent transaction log files. The dump restore utility (**ctrdmp**) is the most common case where this is reported. When one running process detects this file, a **TCOL_ERR** error (537) is returned indicating this collision.

Note: It is important to safeguard these files, however only the *S*.FCS* and *D0000001.FCS* files should remain after a normal server shutdown.

File Name Mapping

c-treeACE maintains a mapping of file names to file numbers. This is transient information and stored in the *D0000000.FCS* file.

Delete Node Queue

D0000001.FCS maintains a list of emptied index nodes. These are eventually cleaned up by the delete node thread and remain available for reuse if needed via this queue.

Active Transaction Logs

Information concerning ongoing transactions is saved on a continual basis in a transaction log file. A chronological series of transaction log files is maintained during the operation of the c-treeACE Server. Transaction log files containing the actual transaction information are saved as standard files. They are given names in sequential order, starting with *L0000001.FCS* (which can be thought of as “active c-treeACE Server log, number 0000001”) and counting up sequentially (i.e., the next log file is *L0000002.FCS*, etc.).

The c-treeACE Server saves up to four active logs at a given time. When there are already four active log files and another is created, the lowest numbered active log is either deleted or saved as an inactive transaction log file, depending on how the c-treeACE Server is configured (see inactive transaction logs).

Every new session begins with the c-treeACE Server checking the most recent transaction logs (i.e., the most recent 4 logs, which are always saved as “active” transaction logs) to see if any transactions need to be undone or redone. If so, these logs are used to perform an automatic recovery. When configuring the c-treeACE Server, the odd and even numbered logs can be written to different physical devices. See "Configuring the c-treeACE Server" (page 153).

Checkpoint Files

S00000000.FCS and *S0000000.FCS* are generated during transaction log checkpoints. These files are used to "kick start" recovery and point to known good transaction states.

Inactive Transaction Logs

Transaction log files no longer active (i.e., they are not among the 4 most recent log files) are deleted by default. To save inactive transaction log files when new active log files are created, add the `KEEP_LOGS` configuration option to the server configuration with a positive number indicating the number of logs to keep. In this case, an inactive log file is created from an active log file by renaming the old file, keeping the log number (e.g., L0000001) and changing the file's extension from "*.FCS"* to "*.FCA*." The Administrator may then safely move, delete, or copy the inactive, archived transaction log file.

Temporary Stream Files

The server creates five stream files at startup. These files prevent errors when the operating system has used a large number of file handles and the server needs a stream file. The file names begin with *FREOPEN* followed by a distinguishing character and ending with *.FCS*. These temporary files are used for internal server operations and should automatically be deleted during a normal server shutdown.

Optional Server System Event Log

The c-treeACE Server maintains two optional system files: *SYSLOGDT.FCS* and *SYSLOGIX.FCS*. *SYSLOGDT.FCS* is a c-treeACE data file with a record for each recordable system event. Unlike the *CTSTATUS.FCS* file, the system log files can be encrypted so entries cannot be added, deleted, or modified with a simple text editor, and vendors can add application specific entries to the log. See "Configuring the c-treeACE Server" (page 153) or your vendor's documentation for information on the SYSLOG (page 138) keywords appropriate to your application.

In case of a system failure, be sure to save all the system files (i.e., the files ending with "*.FCS"*). *CTSTATUS.FCS* may contain important information about the failure. When there is a system catastrophe, such as a power outage, there are two basic possibilities for recovery:

- When the power goes back on, the system will use the existing information to recover automatically.
- The Administrator will need to use information saved in previous backups to recover (to the point of the backup) and restart operations.

7.2 Copying c-treeACE Server Controlled Files

WARNING: c-treeACE Server controlled files should only be copied, moved, or deleted when c-treeACE is shut down. Copying, moving, or deleting files while c-treeACE is in operation can lead to unpredictable errors and data integrity concerns and is never advised.

When a file open is attempted, c-treeACE checks if either a file with the same name is open, or if a file with the same unique ID is open. In either case, the match means a physical file open is not required. Instead, the open count for the file is incremented. The unique file ID permits different applications and/or client nodes to refer to the same file with different names, i.e., different drive or path mappings. However, if two different files have the same ID, problems arise because the

second file will 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.

When a file without a matching name matches the unique file ID, c-treeACE attempts to determine if they are actually different files. If so, it automatically generates a new unique ID for the file. In either case, a message to the system console indicates the names of the two files involved. If this information is not critical to your operation, suppress this message by adding the following entry to the c-treeACE configuration file:

```
MONITOR_MASK  MATCH_FILE_ID
```

Server Unique File Detection - NetWork/Remote/UNC file names

As the c-treeACE Server manages file open/close operations for multiple users, it is critical for it to recognize uniqueness of a file. Different users can refer to the same physical file using different file names through aliases, relative paths, device mappings or SUBST commands. Internally, the c-treeACE Server has tests to determine if two files with different names are really different, or are actually the same physical file being accessed with different paths or alias names. If this internal unique file test is not accurate, the c-treeACE Server may attempt to open the files as two separate physically different files. This presents many problems as the c-treeACE Server is then managing two separate caches for the same file - data integrity can no longer be enforced in these situations.

In some cases, when confronted with file names mapped across a network or referred with UNC syntax, such as “*mymachine*\c*mydata**myfile.dat*”, this internal unique file test incorrectly determines two separate files are being addressed, when actually, the same file is being accessed. One user is using a physical name, while another was is using a UNC name. This problem, while uncommon, leads to serious consequences. As such, c-treeACE performs a number of protections.

COMPATIBILITY NO_UNIQFILE

This option disables attempts to determine if two files accessed with different file names (or paths) and which have identical c-treeACE file IDs are actually the same or different files. This support is added in case our tests for uniqueness are somehow incomplete and lead to unintended file ID reassignments. These modifications give c-treeACE the capability to disable the uniqueness test when files are suspected of having the same internal, “unique” 12-byte ID.

COMPATIBILITY EXACT_FILE_NAMES

This option mandates that all files have the exact same file name in order to be opened. If the internal name test determines that the files are in fact the same physical file, it will not allow the file to be opened with a different name. This compatibility keyword does not permit the same file to be opened with different names. If the same file is attempted to be opened with a different name, then error **EXCT_ERR** (642) will be returned.

There is a subtle interaction with the `NO_UNIQFILE` keyword defined above. The possible outcomes for all the combinations of keywords and files are in the table below. A file is represented by a lower case path, an uppercase name, and a numeric file ID. For example, pA1 has path ‘p’, name ‘A’, and file ID 1. pA1 and qA1 are the same file accessed with different paths;

pA1 and qB1 are different files mistakenly having the same file ID; and pA1 and qB2 are two different files with different file IDs (as expected).

Four possible keyword combinations are possible:

Standard	Neither NO_UNIQFILE nor EXACT_FILE_NAMES
NoUnique	Only NO_UNIQFILE
Exact	Only EXACT_FILE_NAMES
Both	Both NO_UNIQFILE and EXACT_FILE_NAMES

In the outcomes table below, the first (successful) open is for file pA1. The second open is as indicated. In actuality, only the second open for qB1 requires some adjustment or an error return.

Second Open	Standard	NoUnique	Exact	Both
qA1	NO_ERROR	NO_ERROR	EXCT_ERR(642)	EXCT_ERR (642)
qB1	Modify B's file ID and return NO_ERROR	Incorrectly treat B as a shared open of A and return NO_ERROR	Modify B's file ID and return NO_ERROR	EXCT_ERR (642)
qB2	NO_ERROR	NO_ERROR	NO_ERROR	NO_ERROR

The uniqueness test (which is based on system dependent calls) may incorrectly indicate that two files are unique when they are the same. This occurs with certain mappings and/or aliases masking the sameness of the files. If this occurs, the first row of the above table becomes:

Second Open	Standard	No Unique	Exact	Both
qA1	Incorrectly reassign file ID and have same file opened as two different files.	NO_ERROR	Incorrectly reassign file ID and have same file opened as two different files.	EXCT_ERR (642)

The most conservative approach is to turn on both keywords, but of course this requires the same name (and path) to be used for a file on all opens. If the uniqueness test is without weakness, then the standard setting (i.e., neither keyword) works best.

LOCAL FILE TEST

Because of the potential problems with network file names, and because FairCom does not recommend (and discourages) placing c-treeACE Server files or logs on network drives (e.g. drives NOT on the local machine running the c-treeACE Server executable), a warning message is logged to *CTSTATUS.FCS* if a network file is detected. Besides the potential problem for the unique file test, placing data/index or server log files on a mounted network drive will introduce an additional network overhead and jeopardize the server's performance. The WARNING is only issued on the first such occurrence to avoid unnecessary overhead. If either of the `COMPATIBILITY` keywords is active, `NO_UNIQFILE` or `EXACT_FILE_NAMES`, the issue described above is not in play and c-treeACE automatically disables this test.

Due to the possibility of a performance hit, the `COMPATIBILITY NO_TEST_LOCAL` keyword is available to turn off the check of whether a file is local or remote.

7.3 Automatic Recovery

As described in *Starting the c-treeACE Server* (page 35), each time the c-treeACE Server starts it checks the current active transaction logs and determines if any transactions must be undone or redone. If any recovery operation is required, the c-treeACE Server does it automatically. The Administrator need not do anything. The c-treeACE Server displays messages indicating the beginning and the end of the recovery. When automatic recovery completes, the c-treeACE Server is ready to use.

Recovery in Alternate Locations with REDIRECT

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

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

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

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

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

Examples

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

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

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

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

Note: Use double quotes when a filename contains spaces.

Updating IFIL Filenames

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

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

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

Options for Faster Auto-Recovery

The following keywords described in *Advanced Configuration Keywords* (page 329, *http://docs.faircom.com/doc/ctserver/#65772.htm*) reduce the c-treeACE Server disaster recovery time. Reducing the c-treeACE Server auto-recovery time is done at the expense of c-treeACE Server throughput during normal operation. c-treeACE Server auto-recovery is typically a vital consideration in time sensitive, mission critical applications, such as a PBX controller or embedded automation control application. If your application requires the fastest possible data access during normal operations, this section will not be of interest.

```
CHECKPOINT_FLUSH (page 210)
CHECKPOINT_IDLE (page 211)
CHECKPOINT_INTERVAL (page 211)
FORCE_LOGIDX (page 217)
RECOVER_DETAILS (page 229)
RECOVER_FILES (page 229)
RECOVER_MEMLOG (page 230)
TRANSACTION_FLUSH (page 227)
```

7.4 Dynamic Dump

The dynamic dump feature provides an administrator a safe, secure method of backing up data while c-treeACE is operational. The Administrator can schedule a dump of specific files, which may be all files necessary for recovery or a subset of them. The dump executes while c-treeACE is actively processing transactions and is transparent to users.

c-treeACE performs a dump at first opportunity on or after a scheduled time. When beginning a scheduled dump, c-treeACE temporarily halts new transactions and starts the actual dump as soon as all active transactions complete or abort after a predetermined delay period.

More specifically, when a dynamic dump needs to abort a transaction, it puts the transaction into an error state; however it does not actually abort the transaction. With the transaction in an error state, it cannot progress or commit, so the connection that owns the transaction will abort it. Records locked in the transaction will remain locked until the connection aborts the transaction or detects that the transaction has been canceled. If this should occur, it is plausible the application will receive a **TABN_ERR** (error 78), indicating the dynamic dump wait has been exhausted, and the transaction was aborted.

Once the dump commences, transactions can process as usual.

Note: The dynamic dump and recovery processes are intended primarily for files under transaction processing control. Non-transaction controlled files can be dumped with certain restrictions. See "Dump Files Without Transaction Control" (page 113) for more information.

The following sections describe the dump and recovery utilities:

Process	Utility	Explanation
Dynamic Dump	**ctdump**	Dumps data during server operation.
Dump Recovery	**ctrdmp**	Restore files to state as of last dump.
Rollback	**ctrdmp**	Roll the database state to an earlier time following a dump recovery.
Roll Forward	**ctfdmp**	Roll the database state to a later time following a dump recovery.

Scheduling a Dynamic Dump

There are two ways to schedule dynamic dumps:

Server Configuration

The c-treeACE Server configuration file may be used to schedule dynamic dumps. In the configuration file, the keyword DUMP is followed by the name of the script file defining the dump. The path to this script is relative to the server's working directory.

Dynamic Dump Utility

The dynamic dump utility, **ctdump**, is a separate utility for the Administrator to use at any time while the server is active.

To schedule an ad hoc dynamic dump with **ctdump** use the following procedure:

1. While c-treeACE is running, start the utility program **ctdump** as any other program in the environment.
2. Enter the password for the ADMIN administrator account.
3. Enter the current c-treeACE Server name (if assigned a different name than the default). See *Basic Configuration Options* for information on *SERVER_NAME*.
4. Enter the name (with path if necessary) of the dynamic dump script file.

The c-treeACE Server confirms that it has scheduled the requested dynamic dump.

Once a dynamic dump has completed, files may be used for Dump Recovery and/or Rollback.

ctdump - Dynamic Dump Utility

The user may pass a User ID, Password, Dump Script name, and Server Name to a **ctdump** utility, which schedules a dynamic dump. The syntax is as follows:

```
ctdump [adminuser adminpass] dumpscript [servername]
```

- *adminuser*: ADMIN group User ID
- *adminpass*: Administrator password

- *dumpscript*: Name of the dynamic dump script file on the server system. A path relative to the server system may be included.
- *servername*: Optional server name.

For options available when scripting a dynamic dump (added in c-treeACE V11), see *Scripting a Dynamic Dump* (page 100).

The following demonstrates example usage of this utility:

```
ctdump ADMIN ADMIN thescript FAIRCOMS
```

The following error codes are related to dynamic dump operations:

Error Name	Error Code	Explanation
FUNK_ERR	13	Cannot determine file type. Possibly a c-tree V4.3 file?
READ_ERR	36	Failed to read file, either a corrupted or non-tree file.
TCOL_ERR	537	Transaction log collision. Two sets of transaction logs in the same directory?
FCPY_ERR	796	Immediate dump restore file copy failed.
DRST_ERR	797	Immediate dump restore failed.

Scripting a Dynamic Dump

In V11 and later, the Dynamic Dump can send a script to the server and receive a dump stream and/or status messages from the server. The following capabilities are available for scripting a dynamic dump:

1. When scheduling a dynamic dump, the client can send the dump script to the server. Prior to V11, the only option was to create the dump script on the server beforehand and the client passed the name of the existing dump script file to the server.
2. When running a dynamic dump, the client can request that status messages be sent to it while the dump is performed and/or the dump stream file can also be sent to the client process.

To use these options, call the function **dyndmpsetopt()** before calling **dyndmp()**:

```
extern NINT dyndmpsetopt(NINT option,pVOID value);
```

The following are the supported options. All options are disabled by default.

- *DDOPT_SENDSCRIPT* - Send dump script to server. Set value to the script name, or set it to NULL to disable this option. Example:

  ```
  dyndmpsetopt(DDOPT_SENDSCRIPT, ""script.txt"");
  ```

- *DDOPT_RECVSTREAM* - Receive dump stream from server. Set value to YES to enable this option or NO to disable this option. Example:

  ```
  dyndmpsetopt(DDOPT_RECVSTREAM, (pVOID) YES);
  ```

- *DDOPT_RECVSTATUS* - Receive status messages from server. Set value to YES to enable this option or NO to disable this option. Example:

  ```
  dyndmpsetopt(DDOPT_RECVSTATUS, (pVOID) YES);
  ```

- *DDOPT_SETCALLBK* - Set callback function. Set value to the callback function pointer, or set it to NULL to disable the use of the callback function. Example:

  ```
  extern ctCONV NINT mycallback(pVOID pctx,pVOID pdata,NINT datalen,NINT opcode);
  dyndmpsetopt(DDOPT_SETCONTEXT, &mycallback);
  ```

- *DDOPT_SETCONTEXT* - Set callback function context pointer. Set value to the context pointer that will be passed to the callback function. Example:

  ```
  mystruct mycontext;
  dyndmpsetopt(DDOPT_SETCONTEXT, &mycontext);
  ```

- *DDOPT_SETBUFSIZ* - Set communication buffer size. Set value to the buffer size to use. Example:

  ```
  dyndmpsetopt(DDOPT_SETBUFSIZ, (pVOID) 100000);
  ```

Notes:

1) The dump options remain in effect for all dynamic dumps performed by the current connection until they are changed.

2) When the *DDOPT_RECVSTREAM* or *DDOPT_RECVSTATUS* options are used, the following dynamic dump script options are ignored:

COPY_NONCTREE - Non-ctree files cannot be copied.

DATE and TIME - No scheduling of dump for later time.

EXT_SIZE - Only one dump extent is created.

FREQ - No repeat of dump.

SEGMENT - No segmenting of dump stream.

In V11 and later, the **ctdump** utility supports these features through command-line options:

```
usage: ctdump  [-s svn] [-u uid] [-p upw] [-t script] [-b bufsiz] [-n] [-c] [-o backup]
```

Options:

- *-s svn* - c-tree Server name
- *-u uid* - User name
- *-p upw* - User password
- *-t* - Dump script name
- *-b bufsiz* - Use buffer size of *bufsiz* bytes
- *-c* - Send dump script from client
- *-n* - Send progress notifications to the client
- *-o backup_filename* - Write dump stream from server to file on client

Example:

```
# ctdump -u ADMIN -p ADMIN -o backup.fcd -c -t script.txt -s FAIRCOMS -n
```

Results:

```
c-treeACE(tm) Version 11.1.0.46197(Build-150826) Dynamic Backup Utility
Copyright (C) 1992 - 2015 FairCom Corporation
ALL RIGHTS RESERVED.
```

```
Reading dump stream from server with buffer size of 100000
 Start dump. Estimated dump size:      2691072
FAIRCOM.FCS
 86% 100% [     2328576 of      2326528 bytes]
SYSLOGDT.FCS
 89% 100% [       86016 of        81920 bytes]
SYSLOGIX.FCS
 99% 100% [      266240 of       262144 bytes]
S0000000.FCS
100% 100% [        4096 of          128 bytes]
S0000001.FCS
100% 100% [        4096 of          128 bytes]
L0000001.FCS
100% 100% [        2048 of          679 bytes]
   End dump.    Actual dump size:      2705408
Dynamic Dump has been successfully written to the file backup.fcd.
```

Dynamic Dump Options

The FairCom dynamic dump provides a variety of options. These options are included in a script file. This section describes the script file and lists the script keywords and arguments available for defining a dynamic dump.

Dynamic Dump Script File

The dump script file is a plain text file that specifies your options. For example, when to perform a dump, what interval to repeat a dump, and what files to include in a dump.

Format

The script file consists of a series of instructions, each of which is given by a keyword followed by a space and an argument (e.g., the keyword *!DAY* followed by the argument “WED”). All script keywords begin with an ‘!’ and are not case sensitive (i.e., *!DAY* = *!Day*). Arguments are strings of letters, numbers, and punctuation, in the format shown below for each keyword (e.g., WED). New lines divide script keywords and arguments. Keep each keyword/argument pair on a separate line, as in the sample script shown after the list of keywords.

With the following two exceptions, the order of keywords does not matter:

- The next to last script keyword must be *!FILES*, followed by an argument which is a list of the files to be dumped **one file name per line**. Do NOT include a file name on the same line after the *!FILES* keyword.

The last script keyword in the script file must be *!END*, which takes no argument.

Example

```
!TIME   23:00:00
!DAY    Sun
!DUMP   SYSTEM.BAK
!DELAY  600
!FREQ   168
!FILES
FAIRCOM.FCS
!END
```

This script schedules a weekly dump, at 11:00 PM on Sunday nights. The only file included in the dump is *FAIRCOM.FCS*. The system will wait until 11:10 PM (i.e., 600 seconds delay, after the starting time of 11:00 PM) for all active transactions to complete and then it will abort any transactions still active and begin the actual dump. The dump data will be saved in the file named *SYSTEM.BAK*.

Note: The c-treeACE Server configuration file can also control the way lingering transactions are aborted.

Note: The server opens the dynamic dump script when the dump is scheduled, reads its contents, then closes it. At dump execution time the server opens the script again, reads the contents and then closes it before proceeding to dump the files.

!BLOCK_RETURN

Forces **ctdump** to wait until the dynamic dump is completed before returning. Without *BLOCK_RETURN*, **ctdump** returns as soon as the c-treeACE Server receives the dump request. By waiting for completion, *BLOCK_RETURN* permits the System Administrator to determine when the dump is completed. Developers may find it useful to alert a System Administrator when a dynamic dump is complete.

!COMMENT

Default: Off

Informs the c-treeACE Server that the remainder of the script file is for documentation purposes only and is not to be evaluated. Do not place keywords after this keyword.

!COPY_NONCTREE

To include non c-tree files in a dynamic dump, use the dump keyword `!COPY_NONCTREE`. Any file included in the `!FILES` section of the c-treeACE Server dynamic dump script that receives error **FUNK_ERR** (13) or error **READ_ERR** (36) on c-tree open will be treated as a non c-tree file and copied directly to the dump stream base directory. More accurately, to the dump stream base directory plus any subdirectory included in the file's name.

If the destination directory does not exist, the c-treeACE Server will attempt to create it. If this directory creation fails a **FCPY_ERR** (796) is reported.

Note: A check is not made that wildcard specifications in the c-tree/non-ctree file sections match the same filename. In this case, the c-tree file is included in the dump and then the non-ctree file is also copied.

See Also

- !NONCTREEFILES (page 107)

!DATE <mm/dd/yyyy>

Date to perform the dynamic dump or rollback. If the date has already passed, the *!FREQ* interval is applied to find the next scheduled time. If no *!DATE* or *!DAY* is specified, today is assumed.

!DAY <day of week>

Instead of a date, a day-of-week may be used to schedule the dump. They are specified as SUN, MON, TUE, WED, THR, FRI, or SAT. If no date, time or day-of-week is specified, then the dump is scheduled for immediate execution.

!DELAY <seconds>

Number of seconds to wait until aborting active transactions.

If zero, the c-treeACE Server will not abort active transactions. The dump waits indefinitely until all active transactions have completed and no new transactions will be permitted to begin.

If this delay value is greater than zero, the c-treeACE Server waits until either the delay has expired or all active transactions have completed. At this point, it begins the dynamic dump and permits new transactions to start up. If all transactions have not completed, the c-treeACE Server aborts those transactions still in progress, with one of two error messages:

- **TABN_ERR** (78), indicates the transaction has been abandoned.
- **SGON_ERR** (162), a generic error indicating a break in communication between the c-treeACE Server and the application.

!DUMP <dump file>

The name of the file or device into which all the data for all the dump files will be stored.

Note: If a file by this name already exists, it will be deleted at the beginning of the dump and the new dump file replaces the old file.

Note: There must be sufficient space for the dump file, which is limited to the maximum file size for the operating system (2 GB on some systems). If enough space is not available, the dump fails. A failure due to insufficient disk space will not corrupt anything, but additional space must be allocated before a dynamic dump is completed.

!END

Terminates the instructions in the script file. Place *!END* immediately after the *!FILES* keyword and list of files. *!END* takes no argument.

!ENDSEGMENT

Terminates the list of segments when specifying individual segment size and location.

!EXT_SIZE <bytes | NO>

Change the default extent (segment) size from 1GB or disable with NO. See Section Dump To Multiple Files - No Size Limit (page 114) for more details.

!FILES

The *!FILES* keyword is followed by names of files to include in the dynamic dump or rollback. This must be the next to last keyword in the script file and it takes no arguments.

Filenames must begin following the !FILES keyword line with one line for each file. File names should not be listed on the same line as the !FILES keyword. The !END keyword terminates the list of files on a single line.

We strongly suggest that *FAIRCOM.FCS* be included in your list.

Members of a superfile cannot be individually “dumped.” The entire superfile must be dumped; that is, the name of the host superfile, not a superfile member, is placed in the list of files.

The * and ? wildcards are supported.

See *!RECURSE* for other options.

See also:

- *Wildcard Support for File Names* (page 105)
- *Files NOT to Include in Your Dynamic Dump Backup* (page 106)
- *Non-ctree Files Included in a Dynamic Dump* (page 112)
- !COPY_NONCTREE (page 103)
- !NONCTREEFILES (page 107)

Wildcard Support for File Names

Dynamic dump backup and restore scripts specify the names of c-treeACE data and index files that are to be backed up or restored, delimited by the *!FILES* and *!END* script keywords. It is possible to specify filenames using the typical asterisk (`*`) and question mark (`?`) wildcard symbols. See **c-treeACE Standard Wildcards** (page 156).

In addition, the dynamic dump script keyword *!RECURSE* controls directory recursion when using wildcards. The *!RECURSE* keyword only applies when processing a *!FILES* entry containing a wildcard. Keep in mind that it is possible to specify standard file names and wildcard file names, one after the other, in the script. For example:

```
!RECURSE YES
!FILES
myfile.dat
cust*.dat
thedata.dat
*.idx
!END
```

There are three parameters for the *!RECURSE* keyword:

!RECURSE NO	Do not recurse subdirectories.
!RECURSE YES	Recurse underlying directories (max depth 32).
!RECURSE MATCH_SUBDIR	File names and directory names must match.

In the case of *MATCH_SUBDIR*, not only does the file name require a match on the wildcard pattern, but only directory names which match the pattern will be considered for recursion.

The dynamic dump is specifically designed to address c-treeACE data and index files, including superfiles. Please keep in mind: it is possible for your wildcard representation to represent non-c-tree files (see *Non-ctree Files Included in a Dynamic Dump* (page 112)). The following definitions cause all files within the server's `LOCAL_DIRECTORY` to be considered. If any non-ctree files are encountered, the dynamic dump rejects them and a message is written to the *CTSTATUS.FCS* file if the `DIAGNOSTICS DYNDUMP_LOG` keyword is active. A rejection does NOT cause the dump to terminate. It will proceed to the next file.

```
!FILES *.*              !FILES *
!END                    !END
```

Please remember that the dynamic dump does not support individual superfile member names. Specify the host superfile name in the script to back up the members. Here are examples of wildcard names:

- the pattern *m*t* matches *mark.dat* but does not match *mark.dtx*
- the pattern **dat* matches *markdat* and *mark.dat*
- the pattern **.dat* only matches *mark.dat* (not *markdat*)

Files NOT to Include in Your Dynamic Dump Backup

Certain c-tree housekeeping files should not be included in your dynamic dumps. When restoring dumps with these files in them, you may find you end up with **DCRE_ERR** (444) errors as these files collide with housekeeping files of the restore process itself. The following files should be excluded from your list of files to back up:

- *L*.FCS* (Transaction Log files)
- *I*.FCS*
- *S0000000.FCS* (Transaction start file)
- *S0000001.FCS* (Transaction start file)
- *D*.FCS*

WARNING: Don't use *.FCS in your file list.

Exceptions:

FAIRCOM.FCS - Maintains user and group security information. ALWAYS back up this file.

SEQUENCEDT.FCS - Sequence number pool and index. If using the sequence number feature this file is a must to back up.

SEQUENCEIX.FCS

SYSLOGDT.FCS - System logs. If using this feature, consider backing up this file.

SYSLOGIX.FCS

!FREQ <hours>

Hours between successive dumps. For example, 24 to repeat the dump once a day, or 168 to repeat the dump once a week.

!IMMEDIATE_RESTORE

The c-treeACE Server dynamic dump script file keyword, *!IMMEDIATE_RESTORE* instructs the dynamic dump to be immediately followed by a dump restore. This allows transaction-consistent files to be available immediately in a native file system form as opposed to embedded files in the dynamic dump stream file.

A key issue is where the dynamic dump restore utility, **ctrdmp,** can run as it cannot run in the current server directory. If this occurs, error **TCOL_ERR** (537) results indicating that **ctrdmp** conflicted with an existing server operation.

The natural solution is to run **ctrdmp** in the directory that receives the dump stream file, which is called the dump stream base directory. In essence, this requires that *!DUMP <streamFileSpec>* use a file name including a path where the dump restore should run. For example, a dynamic dump script entry of the form

```
!DUMP    I:\dump\mydumpstream
```

will cause the dump stream file *mydumpstream* to be created in the dump stream base directory *I:\dump*. If *!IMMEDIATE_RESTORE* is part of the dump script, then the automatically launched **ctrdmp** is also executed in the *I:\dump* directory.

It is recommended to launch **ctrdmp.exe** from a batch file called *ctrdmp*, which can reside in the server directory. The executable can reside elsewhere (i.e., in the dump stream base directory) and the batch file can call it using a path. The batch file can also do cleanup before (and after) a restore takes place, such as archiving from a prior restore.

Upon restoration of files, the enhanced dump restore will also automatically create any required directory hierarchies for previously backed up files. If an immediate restore operation fails, the server sets the error code for the dynamic dump operation to **DRST_ERR** (797, immediate dump restore failed).

!NONCTREEFILES

A dynamic dump script also supports listing specific files to be backed up as non-ctree files. If the `!FILES` list contains the `!NONCTREEFILES` keyword, all files following that keyword are treated as non c-tree files. Wildcard specifications are allowed. The `!NONCTREEFILES` keyword must appear after the `!FILES` keyword.

Also see the alternative method `!COPY_NONCTREE`

Note: The `!NONCTREEFILES` script keyword does not require specifying the `!COPY_NONCTREE` option in the script.

See Also

- !COPY_NONCTREE (page 103)

!PROTECT and !PROTECT_LOW

The keyword *!PROTECT*, without an argument, added to a dynamic dump script file suspends updates to each non-transaction file while it is being dumped. This ensures the file's data integrity. **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.** With transaction files, the files are consistent because transaction log entries are used to bring all files back to the same point in time, i.e., the effective dump time. In most situations it is more efficient to dump only the data files and rebuild to recreate the indices.

The update suspension is enforced only at the ISAM level unless the keyword *!PROTECT_LOW* is used instead. *!PROTECT* and *!PROTECT_LOW* are mutually exclusive options. The last one in the script is used. FairCom suggests using the *!PROTECT_LOW* when using low-level function calls.

Whether or not *!PROTECT* or *!PROTECT_LOW* are used, resource updates are suspended at the **AddResource()**, **DeleteResource()**, and **UpdateResource()** entry points.

!RECURSE <YES | NO | MATCH_SUBDIR>

Default is NO. Controls directory recursion when using wildcards. The *!RECURSE* keyword only applies when processing a *!FILES* entry containing a wildcard. In the case of *MATCH_SUBDIR*, not only does the file name require a match on the wildcard pattern, but only directory names which match the pattern will be considered for recursion.

!SEGMENT

See details in "Segmented Dynamic Dump" (page 115).

!TIME <hh:mm:ss>

Time of day, on a 24-hour clock, to perform the dynamic dump or rollback. If the time has already passed, then the *!FREQ* interval is used to find the next scheduled time. If a *!DATE* or *!DAY* is specified without a time, then the time defaults to 23:59:59.

The script requires the use or leading zeros for the hour, minute, and second so that each contains two digits. For example, the valid entry for 6:00 is:

```
!TIME 06:00:00
```

The following is *not* a valid entry (notice the single digit, "6," for hours):

```
!TIME 6:00:00
```

If no time, day, or date is specified the dump begins immediately.

Enable Replication During Dynamic Dump Hot Backups

In V11 and later, you can enable replication on one or more files during your dynamic dump "hot" backup operation. This allows an easy way to begin syncing files between systems.

A dynamic dump script can specify the `!REPLICATE` option in the `!FILES` section to instruct the dynamic dump to enable replication support for the files whose names follow the `!REPLICATE` option. Replication is commenced after the dynamic dump has achieved a quiet transaction state. As are other files listed in the `!FILE` section, these files are also backed up by the dynamic dump. For example, the following script will cause the dynamic dump to enable replication for the file *test2.dat* if it does not already have replication enabled:

```
!DUMP mybackup.fcd
!FILES
FAIRCOM.FCS
test1.dat
test1.idx
test2.idx
!REPLICATE
test2.dat
!END
```

If enabling replication fails for any file, the dynamic dump logs an error message to *CTSTATUS.FCS* and terminates. Possible causes of such an error include:

1. Specifying the name of a non-ctree file or a file that does not meet the requirements for replication after the !REPLICATE keyword.
2. If a file is open in exclusive mode at the time of the dump, the dynamic dump is not able to enable replication for the file.

c-tree Files to Include in a Dynamic Dump

A c-treeACE SQL dictionary is composed of several files. You will need to back up all of these files if you want to be able to restore the entire SQL dictionary from your backup. By backing up the correct set of files, you will be able to do a full restore and have your SQL dictionary ready-to-go.

The following files need to be backed up if you want to be able to restore the entire SQL dictionary:

- *FAIRCOM.FCS*
- *ctdbdict.fsd*
- **.dat* in the *ctreeSQL.dbs* folder
- **.idx* in the *ctreeSQL.dbs* folder
- *ctreeSQL.fdd* in *ctreeSQL.dbs\SQL_SYS*

The `!FILES` (page 105) section of your dynamic dump script will look like this:

```
!FILES
FAIRCOM.FCS
ctdbdict.fsd
ctreeSQL.dbs\*.dat
ctreeSQL.dbs\*.idx
ctreeSQL.dbs\SQL_SYS\ctreeSQL.fdd
!END
```

More generally, the following files are FairCom internal files that need to be included in backups to allow recovery to function without `SKIP_MISSING_FILES YES` (in the event these files are changed during the backup interval):

- *FAIRCOM.FCS*
- *SYSLOG*.FCS*
- *SEQUENCE*.FCS*
- *DFRKSTATE*.FCS*
- *ctdbdict.fsd*
- **.dbs\SQL_SYS*.fdd*
- *RSTPNT*.FCS*
- *REPLSTATE*.FCS* (created on the target server by the Replication Agent)

Testing the Backup

The following test should demonstrate that you have backed up everything you need:

1. Use the dynamic dump utility, **ctdump** (page 99), to back up your files into *SYSTEM.BAK*.

   The `!FILES` (page 105) section of your dynamic dump script should include the entries shown earlier.

2. Shut down your c-treeACE Server and rename your *C:\FairCom\V10.3.0\winX64\bin\ace\sql\data* folder to a new (unused) folder name, such as *data.old*:

   C:\FairCom\V10.3.0\winX64\bin\ace\sql\data.old

3. Create a new *data* folder and copy the following files to this location:

   ctrdmp.exe

   SYSTEM.BAK

   Your backup script (the text file that contains the `!FILES` section shown above)

4. Run **ctrdmp** (page 116) to restore your files in place.
5. Now start your c-treeACE Server and connect using c-treeACE Explorer. You should be able to see your restored SQL tables.

Dynamic Dump Defer to Improve Overall I/O Performance

When a dynamic dump runs, the disk read and write operations of the backup process can slow the performance of other database operations. c-treeACE supports an option that allows an administrator to reduce the performance impact of the dynamic dump.

The c-treeACE configuration option:

- `DYNAMIC_DUMP_DEFER <milliseconds>`

This option sets a time in milliseconds that the dynamic dump thread will sleep after each write of a 64KB block of data to the dump backup file.

An application developer can also use the c-tree **ctSETCFG()** API function to set the *DYNAMIC_DUMP_DEFER* value. For example, the following call specifies a 10-millisecond DYNAMIC_DUMP_DEFER time:

- **ctSETCFG(** *setcfgDYNAMIC_DUMP_DEFER, "10"* **);**

The *DYNAMIC_DUMP_DEFER* value set by a call to **ctSETCFG()** takes effect immediately, so this API call can be used by administrators to adjust the speed of a running dynamic dump depending on the amount of other database activity.

> **Note:** The maximum allowed *DYNAMIC_DUMP_DEFER* time is 5000 milliseconds, set at compile-time. If a value is specified that exceeds this limit, the *DYNAMIC_DUMP_DEFER* time is set to *DYNAMIC_DUMP_DEFER_MAX*.

The c-treeACE Administrator utility, **ctadmn**, was also updated to support the dump sleep time option to change this value at run time. The "Change Server Settings" menu is available from the main menu of the **ctadmn** utility.

Dynamic Dump Defer Interval for Improved Backup Performance

The `DYNAMIC_DUMP_DEFER` (page 278) option causes the dynamic dump to pause for the specified number of milliseconds each time it writes 64 KB of data to the dynamic dump stream file. For large backups, even the smallest DYNAMIC_DUMP_DEFER value of 1 millisecond adds significant time to the dynamic dump. For example 100 GB = 1600000 * 1 ms. = 1600 seconds of additional time.

An additional keyword, `DYNAMIC_DUMP_DEFER_INTERVAL` (page 279), specifies the number of 64 KB blocks that are written before the `DYNAMIC_DUMP_DEFER` sleep is performed. For example, `DYNAMIC_DUMP_DEFER_INTERVAL` 16 would cause the `DYNAMIC_DUMP_DEFER` sleep to occur after every 64 KB * 16 = 1 MB of data written to the dump stream file.

> **Note**: If a value greater than 5000 is specified for `DYNAMIC_DUMP_DEFER_INTERVAL`, the value is set to 5000. If a value less than 1 is specified, the value is set to 1.

This option can be set by the **ctSETCFG()** API function:

- **ctSETCFG(** *setcfgDYNAMIC_DUMP_DEFER_INTERVAL, "16"* **);**

A new menu option to set this value has been added to option 10 of the c-treeACE Server Administration (**ctadmn**) menu.

Non-ctree Files Included in a Dynamic Dump

Two alternative methods are available in the c-treeACE Server dynamic dump feature to allow ANY file to be backed up.

Specifying non-ctree Files

A dynamic dump script also supports listing specific files to be backed up as non-ctree files. If the `!FILES` list contains the `!NONCTREEFILES` keyword, all files following that keyword are treated as non c-tree files. Wildcard specifications are allowed. The `!NONCTREEFILES` keyword must appear after the `!FILES` keyword.

Non c-tree Files Dynamic Dump Script Example

```
!DUMP backup.dmp
!FILES
*.dat
*.idx
!NONCTREEFILES
*.log
*.txt
*.cfg
!END
```

Alternative Method

To include non c-tree files in a dynamic dump, use the dump keyword `!COPY_NONCTREE`. Any file included in the `!FILES` section of the c-treeACE Server dynamic dump script that receives error **FUNK_ERR** (13) or error **READ_ERR** (36) on c-tree open will be treated as a non c-tree file and copied directly to the dump stream base directory. More accurately, to the dump stream base directory plus any subdirectory included in the file's name.

If the destination directory does not exist, the c-treeACE Server will attempt to create it. If this directory creation fails a **FCPY_ERR** (796) is reported.

Note: A check is not made that wildcard specifications in the c-tree/non-ctree file sections match the same filename. In this case, the c-tree file is included in the dump and then the non-ctree file is also copied.

Non-ctree File Keywords

```
!NONCTREEFILES
!COPY_NONCTREE
```

Note: The `!NONCTREEFILES` script keyword does not require specifying the `!COPY_NONCTREE` option in the script.

See also

- *!COPY_NONCTREE* (page 103)
- *!NONCTREEFILES* (page 107)
- *!FILES* (page 105)
- *Wildcard Support for File Names* (page 105)

Dump Files Without Transaction Control

It is possible to back up data files that are not under transaction control while the c-treeACE Server remains running. Of course, the safest way to perform a complete backup of data and index files while the c-treeACE Server remains running is to ensure that all your files are under transaction control. This way you are sure 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 for one reason or another. In some cases, developers migrating from the c-treeACE Standalone Multi-user model, *FPUTFGET*, to the c-treeACE Server, choose to use the c-treeACE Server in an *FPUTFGET*-like manner. An *FPUTFGET*-like server is defined with the following c-treeACE 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, if files NOT under transaction control are updated while they are being backed up they cannot be backed up in a consistent state.

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. At this point, 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 indices 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.

Automatic Restore of a Dynamic Dump for Files That Are Ready-to-Use

The c-treeACE Server dynamic dump script file keyword, *!IMMEDIATE_RESTORE*, instructs the dynamic dump to be immediately followed by a dump restore. The idea is to allow for transaction consistent files to be available immediately in a native file system form as opposed to embedded files in the dynamic dump stream file.

A key issue is where the dynamic dump restore utility, **ctrdmp,** can run as it cannot run in the current server directory. If this occurs, error **TCOL_ERR** (537) results indicating that **ctrdmp** conflicted with an existing server operation. The natural solution is to run the **ctrdmp** in the directory that receives the dump stream file. We call this the dump stream base directory. In essence, this requires that *!DUMP <streamFileSpec>* use a file name including a path where the dump restore should run. For example, a dynamic dump script entry of the form

```
!DUMP    I:\dump\mydumpstream
```

will cause the dump stream file *mydumpstream* to be created in the dump stream base directory *I:\dump*. If *!IMMEDIATE_RESTORE* is part of the dump script, then the automatically launched **ctrdmp** is also executed in the *I:\dump* directory.

Upon restoration of files, the enhanced dump restore will also automatically create any required directory hierarchies for previously backed-up files. If an immediate restore operation fails, the server sets the error code for the dynamic dump operation to **DRST_ERR** (797, immediate dump restore failed).

Dump To Multiple Files - No Size Limit

The Dynamic Dump backup feature defaults to breaking-up the backup file (stream file) into multiple physical files (segments). This gets around individual file size limits imposed by the host OS (e.g., 2GB for a typical Unix system). Each backup file segment defaults to 1GB. There is no limit on the number of backup files (segments) supported.

Use the !EXT_SIZE keyword to change the segment size at runtime (up to 2000MB) by setting the argument of !EXT_SIZE to the desired number of bytes. Set the argument to NO to disable this feature and limit the dump to one file up to the OS maximum file size.

When a backup stream file is broken into segments, they are named as follows: original.001, original.002, etc, unless the original dump file has a name of the form name.nnn where nnn represent digits. For example, if the original dump file is named *dump.str*, the first additional segment after *dump.str* gets to the extent size will be *dump.str.001*. However, if the original dump file is named *dump.111*, then the first extent will be *dump.112*.

On some systems, the dynamic dump extent names formed from the original dump stream file name by adding .001, .002, etc. are not legal. Therefore the extent name is first checked internally. If it does not work, the original dump stream file name is modified to produce a safe name in one of the following ways:

- Replace name extension, if any, in original with numeric name extensions (.001, .002, etc.).
- If the original name had no name extension, truncate the name to 8 bytes, and add numeric name extensions.
- If the original name had no name extensions and is not more than 8 bytes, use the name *FCSDDEXT.001* for the first dump stream segment, incrementing the numeric name extension as needed.

Segmented Dynamic Dump

The c-treeACE Server and the **ctdump** and **ctrdmp** utilities support dynamic dumping of segmented files and the creation of segmented (stream) dump files.

Segmented dump files are different from the *!EXT_SIZE* feature that automatically breaks the dump file into 1GB 'extents'. Dumping to segmented files allows you to take advantage of huge file support and to specify the files size and location for each dump file segment.

- To dump segmented data or index files, simply list the host (main) file in the *!FILES* list and the segments will be managed automatically.
- To cause the output dump file produced by the dynamic dump itself to be segmented, use these script entries:

```
!SEGMENT        <size of host dump file in MB>
<dump seg name> <size of segment in MB>
...
<dump seg name> <size of segment in MB>
!ENDSEGMENT
```

The host dump file is the file specified in the usual !DUMP entry. Only the last segment in the !SEGMENT / !ENDSEGMENT list can have a zero size specified, which means unlimited size.

For example, assume *bigdata.dat* is a segmented file with segment names *bigdata.sg1*, *bigdata.sg2*, and *bigdata.sg3*, and the index file *bigdata.idx* is not segmented. To dump these files into a segmented dump file, use the script:

```
!DUMPd:\bigdump.hst
!SEGMENT  50
    e:\bigdump.sg1 75
    f:\bigdump.sg2 0
!ENDSEGMENT
!FILES
    bigdata.dat
    bigdata.idx
!END
```

The host dump file is up to 50 MB on volume D:, the first dump file segment is up to 75 MB on volume E:, and the last dump file segment is as large as necessary on volume F:.

Mirrored File Backups

Mirrored files are supported during dynamic dump and dump recovery as follows:

1. If a mirrored file should be opened for use by an application during a dynamic dump, the dump script should contain the "mirrored" name, i.e., the name with the vertical bar ('|'). For example, *sales.dat*|*msales.dat*.
2. If this is not done, and the dynamic dump opens the primary file, because it is not in use, a client opening the primary|mirror combination gets an **MNOT_ERR** (551, file already opened without mirror). To avoid blocking users from gaining file access, open primary files with their mirrors when specified for dynamic dumps.
3. The dump recovery program recreates both the primary and mirror files. It reproduces the primary file, and copies it over the mirror file.

Dump Progress Messages Displayed in Function Monitor

During backup testing, watching the progress of a running dynamic dump can be beneficial. Adding the keyword `DIAGNOSTICS DYNDUMP_LOG` writes low-level progress messages to the *CTSTATUS.FCS* file. If the `FUNCTION_MONITOR YES` keyword is also active, dynamic dump progress information will also be written to the function monitor.

Mask Routine Dynamic Dump Messages in CTSTATUS.FCS

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

```
CTSTATUS_MASK DYNAMIC_DUMP_FILES
```

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

Run Time Configuration

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

Examples

To turn the option on:

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

To turn the option off:

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

Killing a Dynamic Dump

To kill a dynamic dump, simply execute **ctadmn** and list active clients. The dynamic dump will appear with the COMM PROTOCOL set to DYNAMIC DUMP. Now use the kill clients option to terminate the process. This allows a backup procedure to be canceled (killed) after it has been submitted to the c-treeACE Server.

7.5 Dynamic Dump Recovery

In the event of a catastrophic system failure that renders the transaction logs or the actual data files unreadable, it will be necessary to use a dynamic dump or complete backup, to restore data to a consistent, well defined state. This is known as a dynamic dump recovery.

Note: If you make your own system backups when the c-treeACE Server is not in operation, and include the file *FAIRCOM.FCS*, you can restore from that backup in the event of a catastrophic failure.

Running the Recovery Utility

The **ctrdmp** utility provides dynamic dump recovery. This utility is itself a c-treeACE Server (a bound server) so there are important points to observe when running it:

1. Be sure the particular c-treeACE Server undergoing a recovery is **not** running when **ctrdmp** starts. Two c-treeACE Servers operating simultaneously interfere with each other.
2. Because it is a c-treeACE Server, **ctrdmp** generates temporary versions of all system files associated with a c-treeACE Server (i.e., files with the extension "*.FCS*," as described above). Therefore, the dynamic dump file and the **ctrdmp** utility should be moved to a directory that is not in (or under) the c-treeACE Server's *working directory* (page 362). This is so the system files created by the recovery program will not overwrite working c-treeACE 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.

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.

> **Note:** The same script file used to perform the dump can be used to restore the dump. If a forward dump is planned, include the *!FORWARD_ROLL* keyword.

The dump recovery begins automatically and produces a series of messages reporting the progress of the recovery:

- Each recovered, i.e., recreated, file will be listed as it is completed.
- After all specified files have been recovered, a message is output indicating 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, that is, the time the dynamic dump started.
- A message indicating the dump recovery process finished successfully.

> **Note:** If the dynamic dump data was encrypted with Advanced Encryption (for example, AES), then the *ctsrvr.pvf* password file must be present with the master key information to decrypt and play back this data.

ctrdmp - Dynamic Dump Recovery or System Rollback

```
ctrdmp [ dumpscript ]
```

Used to restore dumps created with **ctdump**.

A successful **ctrdmp** completion always writes the following message to *CTSTATUS.FCS*:

```
DR: Successful Dump Restore Termination
```

A failed **ctrdmp** writes the following message to *CTSTATUS.FCS* when **ctrdmp** terminates normally:

```
DR: Dump Restore Error Termination...: <cterr>
```

where *<cterr>* is the error code.

If for some reason **ctrdmp** terminates prematurely (for example, a fatal error causes **ctrdmp** to terminate abnormally), the "Dump Restore Error Termination..." message might not be written to

CTSTATUS.FCS. In that case, **ctrdmp** might have written error messages to standard output or to *CTSTATUS.FCS* before terminating that helps explain the reason for **ctrdmp** terminating prematurely.

Note: A 32-bit **ctrdmp** could fail with **error 75** if run on transaction logs created by a 64-bit c-treeACE Server, which might support more than 2048 connections.

In V11 and later, the **ctfdmp**, **ctldmp**, and **ctrdmp** utilities display the c-treeACE version used to compile them when they are run.

Environment Variable for Advanced Encryption Password

If this utility has advanced encryption enabled, it can read an encrypted password file instead of being prompted to enter the master password. To enable this, set the environment variable `CTREE_MASTER_KEY_FILE` to the name of the encrypted master password file.

See also

- *Maintaining Database Integrity* (*http://docs.faircom.com/doc/ctserver/8373.htm*) in the *c-tree Server Administrator's Guide*

Recovery Script Options

A recovery script, similar to the dynamic dump script, is used with the recovery utility. In general, this is the same script used to make the dynamic dump. (**Hint!** Back up this script file with your c-tree files so it's readily available!)

The following keywords, with arguments in the same format as dump script options, control the recovery process without effecting the dump itself.

!CLNIDXX

Each file restored by **ctrdmp** is has its index nodes that contain residual key level locks and their associated transaction values cleaned. This permits the file's transaction high water mark to be reset to zero.

!DELETE

Default: !SKIP

The opposite of *!SKIP*. It causes an existing file to be deleted and replaced by the recovered file.

!#FCB <number of files>

Default: 30000 files

When restoring a large number of files from a dynamic dump backup with **ctrdmp**, the dump restore can possibly fail with error **FNUM_ERR** (22, file number out of range). Should you encounter this error with a very large number of restored files, consider the *!#FCB* option in your **ctrdmp** script file to increase this value.

Prior to V9, the default was 100, which could lead to error **22** in situations where this number was too low for **ctrdmp**.

!FORWARD_ROLL

Default: No forward roll is performed

If planning to do a forward roll after a dump recovery, this keyword must be in the recovery script. The keyword is ignored during dynamic dump (backup). When present during dump recovery, this keyword causes a transaction start file to be restored with the archive file extension (i.e., *S*.FCA*). Be sure to rename the file from *S*.FCA* to *S*.FCS* before starting the forward roll. See "Running the Forward Dump Utility for System Recovery" for more information on rolling forward.

!PAGE_SIZE <bytes per buffer>

Default: 8192 bytes

The number of bytes per buffer page, rounded down to a multiple of 128.

Note: Required only if the c-treeACE configuration file changes the default page size.

!REDIRECT <old path> <new path>

Default: no redirect

Redirect output dumped from the old path into the new path. See "Define Alternative Restore Destinations" (page 119) for more information.

!SKIP

Default: *!SKIP*

Skip recovery for any file listed under the *!FILES* keyword if the file already exists.

Note: Be aware of the differences between using *!SKIP* with a recovery and with a rollback (see *System Rollback* (page 121)) where it must be used with caution.

Note: Only the files specified by the *!FILES* keyword will be restored. It is not necessary to restore all files contained in the dump.

Define Alternative Restore Destinations

By default, the **dynamic dump restore returns files to their original directory.** This is due to the fact that file paths are included as part of the filenames in the transaction logs.

To change this default behavior, it is possible to “redirect” the destination of files during a dynamic dump restore.

The dynamic dump script used during restore may contain one or more of the following redirection directives:

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

Note: To specify an empty string for one of the *!REDIRECT* arguments use a pair of double quotes (‘’).

The !REDIRECT keyword substitutes the <new path> for <old path> when found for all file that are restored.

This is often necessary when the restored directory is no longer the same as the original file environment. Consider the case of moving files from one server to another with a slightly different directory structure, notably the case in Windows with a different drive identifier, for example C: to D:. It is also useful for developers to obtain a live “snapshot” of a customer’s database and restore it to an alternative destination for testing, debugging or other purposes.

Keep in mind that c-treeACE SQL includes relative paths as part of the filename as referenced in the transaction logs making this necessary when moving date between c-treeACE SQL database directories.

If you find that files are missing after a restore operation, you should retry the restore with the !REDIRECT directive in place to a known good directory location.

Example 1

The following directives cause files that were backed up using absolute names to be restored into the directory temp (relative to the current directory during restore) and files that were backed up from the directory local (relative to the server working directory) to be restored into the absolute directory *\temp\local*:

```
!REDIRECT        \        temp\
!REDIRECT   local\        \temp\local\
```

The following will add *temp*\ to all restored files:

```
!REDIRECT " "     temp\
```

Example 2

The following will strip d: from any restored files starting with d: (or D:):

```
!REDIRECT d:     ""
```

Note: The *!REDIRECT* keyword only affects the restore operation and is ignored when the script is used for the backup process.

Transaction Dependent Files

When transaction logs are used to recover, rollback (**ctrdmp**) or roll-forward (**ctfdmp**), c-treeACE scans the transaction logs to determine active transactions and to open the files that are updated. When a file cannot be opened, execution may terminate, typically with a **FNOP_ERR** (12) error. It is possible to utilize the `SKIP_MISSING_FILES` option to complete the assessment of active transactions and updated files; and the recovery/rollback/roll-forward will complete, possibly skipping operations on files that could not be opened.

However, this is not always the case as some of the files that were skipped may be created or have been renamed (or deleted), and if they are transaction dependent (*TRANDEP*) files, the transaction logs contain sufficient information to permit them to be properly updated. Adding `SKIP_MISSING_FILES` means that non-*TRANDEP* files may in fact be skipped even though they should have been present.

To avoid requiring `SKIP_MISSING_FILES` when *TRANDEP* files are in use, a new default (V9.1.1) behavior effectively treats *TRANDEP* files as though `SKIP_MISSING_FILES` is turned on, however, for files without *TRANDEP* activities, recover, rollback, or roll-forward may still terminate execution if unexpected missing files are encountered.

This behavior can be turned off by adding the `COMPATIBILITY NO_AUTO_SKIP` configuration keyword to the c-treeACE configuration file, *ctsrvr.cfg*.

Note: It is possible that an unexpected **FNOP_ERR** error can still occur for a *TRANDEP* file, however, this change should greatly reduce the number of unexpected **FNOP_ERR**'s.

7.6 System Rollback

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 under way, 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-treeACE Server.

A rollback, like recovery, involves a dynamic dump script with different keywords to control how the rollback is to be done.

Running the Rollback Utility

The **ctrdmp** utility performs a system rollback as well as dynamic dump recovery. **ctrdmp** checks the first keyword in the script file. If the first line is *!ROLLBACK* the script is used for a rollback. If it isn't, the script is considered a dynamic dump script and used for a dump or a recovery.

Note: As in dump recovery, be sure the particular c-treeACE Server undergoing the rollback is not running when **ctrdmp** starts, since **ctrdmp** is a c-treeACE Server and the two c-treeACE Servers operating simultaneously interfere with each other. Typically, error **TCOL_ERR** (537, transaction log collision) is observed under these conditions.

Perform a rollback as follows:

1. Collect **ctrdmp**, the transaction log files covering the period from the target time to present, and the current log files into a working directory.
2. Start **ctrdmp** the same way as any program in the environment.
3. When prompted, enter the name of the rollback script file to be used.

The rollback begins automatically and produces a series of messages reporting recovery progress. A message returns when the utility completes a successful rollback.

A successful **ctrdmp** completion outputs the following message to *CTSTATUS.FCS*:

```
DR: Successful Dump Restore Termination
```

A failed **ctrdmp** outputs the following to *CTSTATUS.FCS*:

```
DR: Dump Restore Error Termination...: <cterr>
```

where *<cterr>* is a c-tree error code number.

If for some reason **ctrdmp** terminates prematurely (for example, a fatal error causes **ctrdmp** to terminate abnormally), the “Dump Restore Error Termination...” message may not be written to *CTSTATUS.FCS*. In that case, **ctrdmp** may have written error messages to standard output or to *CTSTATUS.FCS* before terminating that explains the premature termination.

Script File for Rollback

The format of the Rollback script is the same as a dynamic dump script. Accepted rollback options are as follows:

!COMMENT

Default: Off

Informs the c-treeACE Server that the remainder of the script file is for documentation purposes only and is not to be evaluated. Do not place keywords after this keyword.

!DATE <mm/dd/yyyy>

Date to perform the dynamic dump or rollback. If the date has already passed, the *!FREQ* interval is applied to find the next scheduled time. If no *!DATE* or *!DAY* is specified, today is assumed.

!#FCB <number of files>

Default: 30000 files

When restoring a large number of files from a dynamic dump backup with **ctrdmp**, the dump restore can possibly fail with error **FNUM_ERR** (22, file number out of range). Should you encounter this error with a very large number of restored files, consider the *!#FCB* option in your **ctrdmp** script file to increase this value.

Prior to V9, the default was 100, which could lead to error **22** in situations where this number was too low for **ctrdmp**.

!FILES

The *!FILES* keyword is followed by names of files to include in the dynamic dump or rollback. This must be the next to last keyword in the script file and it takes no arguments.

Filenames must begin following the !FILES keyword line with one line for each file. File names should not be listed on the same line as the !FILES keyword. The !END keyword terminates the list of files on a single line.

We strongly suggest that *FAIRCOM.FCS* be included in your list.

Members of a superfile cannot be individually “dumped.” The entire superfile must be dumped; that is, the name of the host superfile, not a superfile member, is placed in the list of files.

The * and ? wildcards are supported.

See *!RECURSE* for other options.

See also:

- *Wildcard Support for File Names* (page 105)
- *Files NOT to Include in Your Dynamic Dump Backup* (page 106)
- *Non-ctree Files Included in a Dynamic Dump* (page 112)
- !COPY_NONCTREE (page 103)
- !NONCTREEFILES (page 107)

!ROLLBACK

!ROLLBACK must be the first entry in the script. It takes no argument. If *!ROLLBACK* is not the first entry, the script is interpreted as a dynamic dump script.

!SKIP

Default: *!SKIP*

Skip recovery for any file listed under the *!FILES* keyword if the file already exists.

Note: Be aware of the differences between using *!SKIP* with a recovery and with a rollback (see *System Rollback* (page 121)) where it must be used with caution.

Note: Only the files specified by the *!FILES* keyword will be restored. It is not necessary to restore all files contained in the dump.

!TIME <hh:mm:ss>

Time of day, on a 24-hour clock, to perform the dynamic dump or rollback. If the time has already passed, then the *!FREQ* interval is used to find the next scheduled time. If a *!DATE* or *!DAY* is specified without a time, then the time defaults to 23:59:59.

The script requires the use or leading zeros for the hour, minute, and second so that each contains two digits. For example, the valid entry for 6:00 is:

```
!TIME 06:00:00
```

The following is *not* a valid entry (notice the single digit, "6," for hours):

```
!TIME 6:00:00
```

If no time, day, or date is specified the dump begins immediately.

7.7 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-treeACE 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 (as well as !REDIRECT (page 125)). 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-treeACE 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.

Note: If the dynamic dump data was encrypted with Advanced Encryption (for example, AES), then the *ctsrvr.pvf* password file must be present with the master key information to decrypt and play back this data.

Forward Roll File Redirection

While restoring from a backup with the forward roll utility, **ctfdmp** supports applying file name redirection rules to the file names that appear in the transaction logs. To use this feature, run **ctfdmp** with the *!REDIRECT* option, specifying the name of a text file that contains the redirection rules. For example, the following command indicates that the file *redir.txt* contains redirection rules:

```
ctfdmp !REDIRECT redir.txt
```

Each line in the file contains the portion of the file name to replace and its replacement. Place double quotation marks around a string if it contains spaces. A line that begins with a semicolon is ignored.

Examples

To replace an empty path with *output*\:

```
;Replace empty path with output\
"" output\
```

To replace *Program Files(x86)* with *Program Files*:

```
"Program Files(x86)" "Program Files"
```

To replace *production*\ with *test*\:

```
production\ test\
```

7.8 Transaction Log Dump

A transaction log dump is not something a Server Administrator typically needs to use, but we explain it here to be complete. Developers most often use this functionality as an aid to design, code, and debug an application being developed for use with a c-treeACE Server.

ctldmp is a utility providing a partial dump of transaction log files. This utility will attempt to create an ASCII log from the records in the transaction log and display it on the screen. It converts only the first 39 bytes of each record in the transaction log.

In V11 and later, the **ctfdmp**, **ctldmp**, and **ctrdmp** utilities display the c-treeACE version used to compile them when they are run.

Options for Transaction Log Dump

The format of keywords for defining a transaction log dump is the same as the dynamic dump script file, but they are not put in a separate file or script. Instead, they are entered along with the name of the program when starting the program.

The keywords and arguments for **ctldmp**, the transaction log dump utility, are:

DATE <mm/dd/yyyy>

Begin dumping transactions as of the date specified. If no date is specified, begin dumping transactions from the beginning of the log file.

LOG <number>

Dump transactions beginning with the specified log. If no log number is specified, dump all log files meeting all other specifications.

OFF

Value of the position entry, offset, in the log dump that must be matched for the transaction to be listed. It is the ‘P’ field which follows the transaction number in the dump listing.

POS

Byte position in log to start the dump.

TIME <hh:mm:ss>

Begin dumping transaction as of the time specified. If a date is specified, then the date and time are used in conjunction with each other. If a date is not specified the current date is the default.

TRAN

Transaction number which must be matched in order for the transaction to be listed.

TYPE

Transaction type, which must be matched for the transaction to be listed. The following code numbers correspond to the specified transaction type:

TYPE Value	Explanation
02	Add key value.
04	Delete key value.
07	Begin transaction.
08	Commit transaction.
09	Abort transaction.
12	New record image.
14	Old record image.

TYPE Value	Explanation
15	Difference of old/new record images.
26	Server checkpoint.
27	Open file.
28	Create file.
29	Delete file.
30	Close file.
31	Client login.
32	Client logoff.
34	End of log segment.
40	Abandon transaction.

CHKPNT yes

Outputs detailed information for each checkpoint encountered during a dump. For example, by using the following command line, the log dump begins with log file *L0000100.FCS*; only dumps checkpoints, and lists detailed information about checkpoints. *tran* types are found in *ctopt2.h* and the table above.

```
ctldmp log 100 type 26 chkpnt yes
```

Running a Transaction Log Dump

Note: Like **ctrdmp**, **ctldmp** is itself a c-treeACE Server, therefore, the particular c-treeACE Server that generated the transaction logs being dumped by this utility should not be running while **ctldmp** is running. Typically, error **TCOL_ERR** (537, transaction log collision) is observed under these conditions.

Running a transaction log dump is a one-step process completed by starting **ctldmp** as any program in the environment, followed by up to three keyword/argument pairs specifying date, time and log number. **ctldmp** runs automatically, without prompting for any information, and informs you when it completes the transaction log dump.

ctldmp option to create transaction start files from checkpoints in transaction log files

In V10.3, an option was added to the **ctldmp** utility to create transaction log start files for the transaction logs that it scans. Specify the *csf* ("create start files") option to use this feature.

The start files are named *S<lognumber>_<sequencenumber>.FCA*, where *lognumber* is the transaction log number and *sequencenumber* is the checkpoint number in that log (starting from 1).

These start files can be renamed to *S0000000.FCS* and *S0000001.FCS* so that c-treeACE Server can use the checkpoint positions as starting points for automatic recovery.

Example:

```
# ctldmp log 1254 csf

getlogfil
LOGPOS:1254-002473fax #0:690034176 P0051a057f3-00x  F0000-000 T26 U03 A0000x L139672 042 CHKPNT
0000e0007fa0a000a000920092007300000000071001200e0000000000510000000000000000
300054001cb10000200002042002c00000030008a20f0003400000040004b20100000000000000
........q....."".."".r<.........x.... ..............T.................
Created start file S0001254_0001.FCA for log position 0x002473fa

        Sat May 25 01:19:31 2013

LOGPOS:1254-0026a624x #0:690034177 P0051a057f3-00x  F0000-000 T26 U17 A0000x L10684 042 CHKPNT
ffffe000f720a000a000920092009200000000092001200e0000000000720000000000000000
bfff6400a340000020000200420082000000300c200f000340000004000830010000000000000
.........s$........."".."".."".."".... ..............x#.............
Created start file S0001254_0002.FCA for log position 0x0026a624

        Sat May 25 01:19:31 2013
```

7.9 Controls for Performance AND Safety of Non-Transaction Updates

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

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

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

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

In V11 and later it is possible to define a vulnerability window limiting potential loss of updates for your non-transaction data and index files. c-treeACE Server supports options to write dirty cache pages to the file system within a specified time period. This means that no more than a set amount of time can pass where data is not flushed to disk.

The following c-treeACE Server configuration options set the time limit in seconds that a data cache page or index buffer can remain dirty before it is written to the file system cache. The default time limit is 60 seconds. Specify `IMMEDIATE` to cause dirty pages to be written immediately. Specify `OFF` to disable time limit-based flushing.

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

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

Monitoring Non-Transaction Data Flush

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

Tuning Non-Transaction Data Flush

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

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

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

Non-Transaction Flush Diagnostics

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

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

7.10 Checkpoint Requirements

c-treeACE periodically writes checkpoints to the transaction logs. Automatic recovery, rollback, and forward roll use the most recent checkpoint listed in the transaction start files as the starting point. This section explains the conditions that a checkpoint must meet to be used for these operations.

Forward Roll

A forward roll can only be started from a checkpoint that is logged when **all** of the following conditions are true:

1. No transactions are active.
2. No abort node list entries exist (except if using the **ctrdmp** utility and the forward roll starts from the begin dump checkpoint, in which case it is allowed).
3. No index buffers contain unflushed updates for committed transactions.
4. No data cache pages contain unflushed updates for committed transactions.

Due to these requirements, there is no guarantee that a checkpoint logged by calling **CTCHKPNT()** can be used as the starting point for a forward roll. If a forward roll is attempted from a checkpoint that does not meet these requirements, the forward roll fails with error 510 (**RFCK_ERR**, "active checkpoint at start of forward roll").

Three options are available for generating a checkpoint that can be guaranteed to be usable for a forward roll operation:

1. **Perform a dynamic dump**: This is probably the most commonly-used option to provide a starting point for a forward roll operation. The dynamic dump achieves a quiet transaction state and flushes all updated index buffers and data cache pages for transaction-controlled files. Then it writes a "begin dump" checkpoint to the transaction logs and allows transaction activity to resume. The dynamic dump writes the specified data and index files to the dump stream file, and then it writes an "end dump" checkpoint and copies the transaction logs (the logs containing these two checkpoints and all logs between these two logs) to the dump stream file. When the **ctrdmp** utility is run, it reads the data and index files from the dump stream file and recovers them to their state as of the begin dump checkpoint. If you include the `!FORWARD_ROLL` option in the dump restore script, **ctrdmp** creates a start file that points to the begin dump checkpoint, which can be renamed from an *FCA* extension to *FCS* to serve as the starting point for a forward roll.
2. **Call ctQUIET()**: Call **ctQUIET()** with a mode that ensures that the forward roll transaction state requirements are met. For example, use *ctQTblockALL* | *ctQTflushAllFiles*. While the server is quiesced, make a copy of the data files, index files, and transaction logs. The logs will contain a checkpoint that can be used to roll forward.
3. **Shut down c-treeACE Server cleanly**: Shut down c-treeACE Server cleanly so that all clients disconnect and all files are closed, and c-treeACE Server writes a clean final checkpoint to the transaction log. The message "Perform system checkpoint" in *CTSTATUS.FCS* indicates that the final checkpoint was written. For example:

```
Wed Oct 5 10:04:01 2016
 - User# 00021 Server shutdown initiated
Wed Oct 5 10:04:03 2016
 - User# 00021 Communications terminated
Wed Oct 5 10:04:03 2016
 - User# 00021 Perform system checkpoint
Wed Oct 5 10:04:03 2016
 - User# 00021 Server shutdown completed
```

A final checkpoint (logged at a clean server shutdown) should also have the required attributes. If a checkpoint does not conform to the conditions listed above, a forward roll beginning at such a checkpoint will fail with **RFCK_ERR** (510).

Rollback

Rollback can be started from any checkpoint. Starting with a point-in-time copy of the data files, index files, and transaction logs (acquired by using dynamic dump or **ctQUIET()** for example), rollback begins with the most recent checkpoint listed in the transaction start files. First, automatic recovery is performed to bring the files up to the state of the last committed transaction in the transaction logs, and then rollback undoes operations back to the requested point-in-time.

Calling CTCHKPNT

Although rollback can use any checkpoint, the checkpoint requirements for forward roll mean that a call to **CTCHKPNT()** is not guaranteed to be usable for forward roll. Note that each time a

checkpoint is logged, the transaction start files are updated, so only the two most recent checkpoints will be listed in the two start files. The start files are used to provide the starting checkpoint position to forward roll and rollback. By calling **CTCHKPNT()** you are simply updating the start files more frequently. The two start files will never refer to more than two checkpoints at a time, and c-treeACE automatically writes checkpoints to the transaction logs periodically (typically at least three checkpoints per log).

Also remember that a forward roll or rollback requires more than just a starting checkpoint: the state of the data and index files must correspond to the current state of the transaction logs and the position of the starting checkpoint. To roll forward or back, you will need to have saved a point-in-time copy of the data files, index files, and transaction logs. Unless you save off this complete set of files when you call **CTCHKPNT()**, that checkpoint will not be useful in rolling forward or rolling back.

8. Monitoring c-treeACE

The c-treeACE Server V8.14 and later tracks and reports a wealth of performance-oriented statistics. The server's Performance snapshot capability enables the capture of performance monitoring data using a combination of configuration file options and the SnapShot c-treeACE API function. The performance data can be captured automatically at specified intervals or on demand in the following ways:

- The **ctstat** utility provides a command-line interface to the *SnapShot* API function, supporting output of c-treeACE Server statistics at the c-treeACE Server system, user, file, and function levels.
- Configuration file options support automatic performance snapshots by specifying the interval between automatic snapshots and the contents of the snapshots. The performance data captured by automatic snapshots are written to the c-treeACE Server system event log (*SYSLOG*) files.
- Use `DIAGNOSTICS` options to capture the automatic system snapshot data to the human-readable *SNAPSHOT.FCS* file.
- The SnapShot API function can control automatic snapshots, overriding configuration options (if any), and can capture on-demand performance data:
  - to either the *SYSLOG* files or to the human-readable *SNAPSHOT.FCS* file, or
  - as a return to the calling program.

The following sections discuss how to use the performance monitoring abilities of the c-treeACE Server.

8.1 Performance Monitoring Using the ctstat Utility

The **ctstat** utility is a client utility used to display statistics collected by the c-treeACE Server. It is found in the client folder of the server installation and demonstrates the use of the SnapShot function. This section describes the reports that this utility produces and how to use the utility to generate these reports.

See Also

- Statistics Monitoring Utility - ctstat (page 62)

8.2 Performance Monitoring Using Server Keywords

This section describes automatic performance snapshot logging using c-treeACE Server configuration keywords.

Automatically Logging Performance Snapshots

The c-treeACE Server supports automatically logging performance snapshots to the *SYSLOG* files and to the human-readable file *SNAPSHOT.FCS*. The following sections describe how to enable automatic snapshots.

Automatic Logging to the Server System Event Log

The `SNAPSHOT_INTERVAL` keyword enables automatic snapshots at specified intervals:

```
SNAPSHOT_INTERVAL  <minutes>
```

By default, only the system snapshot is captured. To add user or file-specific snapshots to the data captured, use one or more of the following configuration entries:

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

Files and users added to the snapshots are said to be activated. Users and files may be activated whether or not the automatic snapshots are turned on in the configuration file. However, the activation has no effect until snapshots are written to the *SYSLOG* files.

The *<user ID>* and *<file name>* arguments may include wildcard matching characters: “*” matches an arbitrary number of any characters, and “?” matches exactly one of any character. A pattern of simply “*” matches any user or file name. For example, the following keywords activate all users, any file ending in “.dat”, and the file *journal.idx*:

```
SNAPSHOT_USERID     *
SNAPSHOT_FILENAME   *.dat
SNAPSHOT_FILENAME   journal.idx
```

User IDs are not case sensitive. File name case sensitivity depends on the platform. For example, Windows is case insensitive and Unix is case sensitive. The file names activated must match the file name used to first open the file. In particular, paths used in the activation list and during the call to open the file must match.

Automatic Logging to SNAPSHOT.FCS

Write system snapshots to the human-readable *SNAPSHOT.FCS* text file with the following `DIAGNOSTICS` options:

```
DIAGNOSTICS   SNAPSHOT_SHUTDOWN
DIAGNOSTICS   SNAPSHOT_AUTOMATIC
```

`DIAGNOSTICS SNAPSHOT_SHUTDOWN` writes a system snapshot to *SNAPSHOT.FCS* at the start of the server shutdown process. `DIAGNOSTICS SNAPSHOT_AUTOMATIC` writes any automatic snapshots to *SNAPSHOT.FCS* instead of to the *SYSLOG* files. However, only the system snapshot is written. Snapshots for activated users and/or files are ignored.

8.3 Performance Monitoring Using the SnapShot API

c-treeACE Server configuration options provide an easy way to enable automatic performance snapshots. Additional flexibility and access to the complete set of statistics maintained by the c-treeACE Server is available programmatically through the SnapShot c-treeACE API function. This API function:

- Controls automatic snapshots, overriding configuration options (if any).
- Writes snapshots to the *SYSLOG* files on demand, whether or not automatic snapshots are active.
- Writes system snapshots in human-readable form to *SNAPSHOT.FCS*.
- Returns snapshot data on-demand to the calling application program.

See Also

- A subset of the SnapShot statistics is available using server keywords. See *Performance Monitoring Using Server Keywords* (page 132).
- For information about using the SnapShot function, see *Performance Monitoring Using the SnapShot API http://docs.faircom.com/doc/ctreeplus/#29622.htm* and *SnapShot* (*http://www.faircom.com/doc/ctreeplus/snapshot.htm*) in the *c-treeACE Programmer's Reference*.

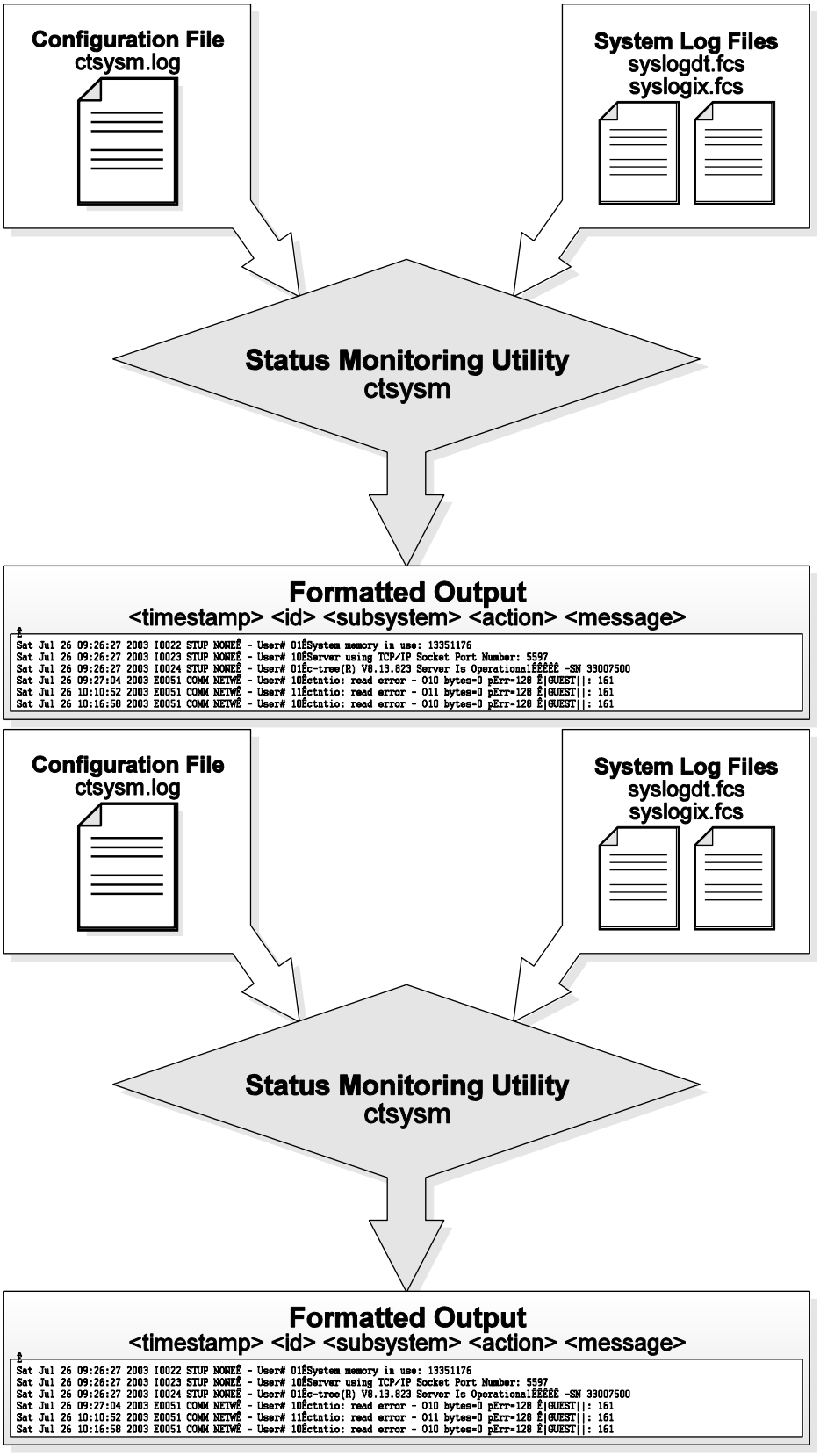
8.4 c-treeACE Server Status Monitoring Utility, ctsysm

The c-treeACE Server status monitoring utility, **ctsysm**, facilitates the monitoring of error, warning, and informational messages logged to the server status log, *CTSTATUS.FCS*, by c-treeACE. Using this utility, the c-treeACE Server status log can be monitored by an automated external processes.

Using a customizable configuration file, the **ctsysm** utility provides:

- A well-defined format for server messages to make them readable by automated systems
- Additional context as to whether the message is an error or purely an informational message
- The c-treeACE Server subsystem that is involved
- A recommended action

The following diagram shows a conceptual view of the operation of the **ctsysm** utility:



Using the ctsysm Utility

To use the **ctsysm** utility, the c-treeACE Server must be configured to log messages to the system log. This can be achieved by adding the following keyword to the server's configuration file:

```
SYSLOG (page 138) CTSTATUS
```

With this keyword in place, the server logs each entry in the *CTSTATUS.FCS* file to the system log files (*SYSLOGDT.FCS* and *SYSLOGIX.FCS*).

The utility can read the system log from the beginning each time it starts up, or it can save its current position and start again from that set position by specifying the -f command line option.

The following is the supported command line usage of the c-treeACE Server Status Monitoring Utility:

```
ctsysm [-s svn] [-u uid] [-p upw] [-r rpt] [-c cfg] [-f fil] [-l log]
```

- *-s svn* - c-treeACE Server name
- *-u uid* - user name
- *-p upw* - user password
- *-r rpt* - repeat interval in seconds (for example, *-r 1* is a one-second delay before checking for new messages)
- *-c cfg* - config file
- *-f fil* - save/restore state to file
- *-l log* - status log name (*SYSLOGDT.FCS*)
- *-e* - direct **ctsysm** error messages to standard output in the same format as messages read from the server status log

Example

The following command causes **ctsysm** to monitor the c-treeACE Server system log for the server FAIRCOMS (*-s FAIRCOMS*) for status messages, checking for new messages every second (*-r 1*), saving the position of the last entry read to the file *ctsysm.log* (*-f ctsysm.log*).

```
ctsysm -s FAIRCOMS -u ADMIN -p ADMIN -r 1 -c ctsysm.cfg -f ctsysm.log
```

The ctsysm utility outputs messages in the following format:

```
<timestamp> <code> <subsystem> <action> <text>
```

Sample output from the utility is shown below (each message is output as a single line but is shown split into two lines here):

```
Thu Jul 29 16:44:04 2004 I0455 STUP NONE                    - User# 01
   Alternative server name...  FAIRCOMS
Thu Jul 29 16:44:04 2004 I0492 STUP NONE                    - User# 01
   Compatibility bit maps: 00000000 00002000x
Thu Jul 29 16:44:04 2004 I0491 STUP NONE                    - User# 01
   Diagnostic bit maps: 00000000 00000000x
Thu Jul 29 16:44:04 2004 I0490 STUP NONE                    - User# 01
   64-bit File Address Support
Thu Jul 29 16:44:04 2004 I0489 STUP NONE                    - User# 01
   6 Byte Transaction Numbers
Thu Jul 29 16:44:04 2004 I0488 STUP NONE                    - User# 01
   NOWAIT usrsema enabled
```

Note: When the repeat (*-r*) option is used, the utility can be stopped by sending it a SIGINT signal.

ctsysm Configuration File

The configuration file, *ctsysm.cfg*, contains server messages classified by the server subsystems that generate the log entries, the recommended actions, and details for each message. The configuration file is simply a text file, an abbreviated copy of which is shown later in this document. The configuration file can be used as-is, although some of the default recommended actions are general and administrators may want to customize these.

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 or application-specific information about specific messages becomes available, without having to update the server executable.

ctsysm Configuration File Sample

Below is a portion of a sample of the current configuration file (*<install path>\source\ctsysm.cfg*):

```
[Version]

c-treeACE Server Status Log Monitor Configuration File V11.0

[Subsystems]

CADM Connection administration
COMM Communication
DIAG Diagnostic information
DDMP Dynamic dump
FMNT File maintenance
.
.
.
```

```
[Actions]

CCFG The server found an unsupported or invalid option in the server configuration file. Check
the server configuration file.
CFGW An option specified in the server configuration file might lead to undesired behavior. Review
the message to understand the situation.
CKDS The operation failed due to insufficient disk space. Free up disk space to allow the operation
to complete.
.
.
.

[Messages]

;================================================
;Server configuration messages
;================================================
F0001 SCFG CCFG 1 Log set range too large
F0002 SCFG CCFG 1 Cannot read existing encrypted logs.  Enable log encryption to proceed.
F0003 SCFG CCFG 1 Bad COMPATIBILITY argument
.
.
.

;================================================
;Server startup messages
;================================================
F0041 STUP SUPT 1 D0000000.FCS:
F0042 STUP SUPT 1 I0000001.FCS
F0043 STUP SUPT 1 System monitor queue creation error
.
.
.

;================================================
;Diagnostic information messages
;================================================
F0135 DIAG SUPT 1 Unexpected internal c-tree(R) error #
E0136 DIAG SUPT 1 relaim_queue processing error
E0137 DIAG SUPT 1 SnapShot event error
.
.
.
```

8.5 Server System Event Log Keywords

c-treeACE optionally maintains a system event log, SYSLOG. This is maintained in two system files: *SYSLOGDT.FCS* and *SYSLOGIX.FCS*. These files comprise a c-treeACE data file and index pair with a record for each recordable system event. Unlike the text based *CTSTATUS.FCS*, SYSLOG can be encrypted such that entries cannot be added, deleted, or modified with a simple text editor, and vendors can log application specific entries.

The System Event Log contents are controlled by `SYSLOG` configuration keywords in *ctsrvr.cfg*, the *ctsrvr.set* settings file, or from the command line. They are entered as pairs in the form of: `SYSLOG <keyword>`. As many of these pairs as desired may be used at the discretion of your application vendor.

Current `SYSLOG` options include:

ADMIN_API	Only allow users in the ADMIN group to use the **SystemLog()** function to create vendor-defined entries in the log.
CTSTATUS	Log each entry to *CTSTATUS.FCS* in the System Event Log, except for those entries which occur before or after the system logging monitor is in operation.
DELETE_FILE	Log file deletes and restores.
DISABLE_API	Do not allow any calls to the **SystemLog()** function for user defined entries.
DYNAMIC_DUMP	Log the beginning and end of dynamic dumps and a result for each file dumped.
ENCRYPT	Encrypt the *SYSLOG* files.
USER_INFO	Log all logons, logoffs, and changes to user logon profiles.
NONE	Used in a settings file to eliminate additional SYSLOG entries in a server configuration file.
LOGFAIL_PURGE	Causes an automatic purge of the oldest entries in the log if the system cannot add a record to *SYSLOGDT.FCS*. All the entries occurring on the oldest day are deleted unless there are only entries for the current day in which case no entries are purged. After a successful purge, an attempt is made to add the new entry that triggered the automatic purge. If this add succeeds, the system log operation continues in its usual fashion.
LOGFAIL_CTSTATUS	If there is no LOGFAIL_PURGE entry in the configuration file, or if the purge fails, the log entries will be rerouted to *CTSTATUS.FCS* if LOGFAIL_CTSTATUS is in the configuration file. This disables SYSLOG CTSTATUS; i.e, no more entries are made to the system log.
LOGFAIL_TERMINATE	If there is no automatic purge or it fails, and if there is no re-routing to *CTSTATUS.FCS*, either the system log will stop operation, or if LOGFAIL_TERMINATE is in the configuration file, the c-treeACE Server will shutdown. **Note:** USE LOGFAIL_TERMINATE WITH CAUTION!

9. Performance Optimization

Performance is a critical concern in many high availability applications. There are many options available with c-treeACE to maintain the highest levels of performance. These can be both from the application development side (client) and on the server side. Choices such as which transaction mode for files, index and data cache sizes, and operations done with those files all interact in complex manners. This section describes some of the outstanding issues surrounding performance and data integrity.

9.1 Options for Advanced Applications

The following items can optimize c-treeACE Server throughput and are intended for use by advanced application requirements.

Caution: The suggestions in "Optimizing Transaction Processing - ADVANCED" can make disaster recovery difficult or impossible.

I/O caching

If the computer running the c-treeACE Server has sufficient memory and the size of the files controlled by the c-treeACE Server are relatively large, increasing `DAT_MEMORY` and `IDX_MEMORY` can potentially improve performance. In general, the larger the data and index cache sizes, the better the performance for high volume environments. The c-treeACE Server uses a hashed caching algorithm, so there is no need for concern with having the cache sizes set too large.

Fastest Platform

A commonly asked question is which c-treeACE Server platform offers the fastest response times. The performance of the c-treeACE Server is largely dependant on the host hardware and the communication protocol. The faster the CPU and disk I/O subsystem, the faster the c-treeACE Server responds. The internal performance differences for the c-treeACE Server across platforms are negligible.

Base the decision for which hardware platform to choose for the c-treeACE Server on the optimum hardware specifications using the following order of priority:

1. Fastest Disk I/O Subsystem
2. Fastest CPU
3. Fastest and most supported RAM

Communication Protocol

The c-treeACE Server supports many communication protocols in addition to many operating system/hardware combinations. Typically, the largest I/O bottleneck with the client/server model is the communications between the server and the clients. Choosing the best suited communication protocol for the database server can play a crucial role in the client side response times. Due to all the variables affecting response times, (record size, quantity of records, number of users, network traffic, speed of the network cards . . .), it is impossible to provide an absolute guideline for which protocol to use. The best way to determine the optimal protocol for a particular platform is to conduct time trials with the available protocols. However, as a rule, the platform's protocols will typically be the fastest.

It is possible for users to use the c-treeACE Server across WANs of varying distances. The performance in this case depends upon several factors such as:

- Record size
- Type of operations being performed
- Distance
- Delay
- Number of different routers and switches it must go through

Other delays that are added to the travel from point A to point B are caused by such things as congestion, mismatched MTUs (Maximum Transmission Units), or other physical issues. These result in an increase in time to send a request to the server and also to receive the resulting message.

In order to minimize such delays, consider the points below:

- Set your machine to the optimal MTU to reduce packet fragmentation as the message passes through various routers.
- Within your application, you can improve performance by using BATCHES where possible to process multiple records.
- Avoid mismatched MTU sizes on different routers and switches since this can cause packet fragmentation adding significant delays to the delivery of the message.

Flexible I/O Channel Usage

A configuration keyword permits more flexible usage of multiple I/O channels. Without this feature, the *ctDUPCHANEL* file mode bit enables a file to use NUMCHANEL simultaneous I/O channels, where NUMCHANEL is set at compile time, and has traditionally been set to two. For superfile hosts with *ctDUPCHANEL*, 2* NUMCHANEL I/O channels are established. Two new server configuration keywords are now available for this feature:

```
SET_FILE_CHANNELS <file name>#<nbr of I/O channels>
DEFAULT_CHANNELS <nbr of I/O channels>
```

- *SET_FILE_CHANNELS:* Permits the number of I/O channels to be explicitly set for the named file regardless of whether or not the file mode, at open, includes *ctDUPCHANEL*. A value of one for the number of I/O channels effectively disables *ctDUPCHANEL* for the file. A value greater than one turns on DUPCHANEL and determines the number of I/O channels

used. The number of I/O channels is not limited by the compile time NUMCHANEL value. You may have as many `SET_FILE_CHANNELS` entries as needed.

- *DEFAULT_CHANNELS:* Changes the number of I/O channels assigned to a file with *ctDUPCHANEL* in its file mode at open, unless the file is in the `SET_FILE_CHANNELS` list. The default number of channels is not limited by the NUMCHANEL value.

Note: When *ctFeatCHANNELS* is enabled, multiple I/O channels are disabled for newly created files. The multiple I/O channels take affect only on an open file call. Also, depending on default number of I/O channels, a superfile host not in the `SET_FILE_CHANNELS` will use no more than 2 * NUMCHANEL I/O channels.

9.2 Transaction Control Options

c-treeACE offers three modes of transaction processing logic:

1. **No Transaction Control**.
   With no transaction control defined for a data file, read/write access to the file will be very quick. However, no guarantee of data integrity will be available through atomicity or automatic recovery.
2. **Preimage Transaction Control** (*PREIMG* - partial) (atomicity only).
   If the *PREIMG* file mode is used, database access will still be fast, and some data integrity will be provided through atomicity. With atomicity only, (*PREIMG*), changes are made on an all or nothing basis, but no automatic recovery is provided.
3. **Full Transaction Control** (*TRNLOG* - atomicity with automatic recovery).
   If your application files have been setup with the *TRNLOG* file mode, all the benefits of transaction processing will be available, including atomicity and automatic recovery. Automatic recovery is available with *TRNLOG* only, because *TRNLOG* is the only mode where all changes to the database are written immediately to transaction logs. The presence of the transaction log (history of changes to the files) allows the server to guarantee the integrity of these files in case of a catastrophic event, such as a power failure. Recovering files without a *TRNLOG* file mode from a catastrophic event will entail rebuilding the files. File rebuilding will not be able to recover data not flushed to disk prior to the catastrophic event.

Note: Atomicity and automatic recovery are defined in "Glossary" (page 362).

It is important for application developers to understand the complete aspects and consequences of any chosen transaction mode.

Transaction Options

SUPPRESS_LOG_FLUSH

Full transaction processing offers maximum data integrity, however, at some expense to performance. Using the `SUPPRESS_LOG_FLUSH` option reduces overhead with transaction processing log file flushes, but at the expense of automatic recovery. **Suppressing the log flush makes automatic recovery impossible.** This keyword is typically considered only with the `PREIMAGE_DUMP` keyword described below.

PREIMAGE_DUMP

Although automatic recovery is not available to *PREIMG* files, it is possible to perform periodic dynamic backups. By using the `PREIMAGE_DUMP` keyword, it is possible to promote *PREIMG* files to full *TRNLOG* files during the server dynamic dump process (see "Dynamic Dump" (page 98)). The promotion to a *TRNLOG* file means a full transaction log (history of the file changes) will be maintained only during the dump process. This guarantees changes made to data while the backup is occurring is saved in these specially maintained transaction logs. The ability to dynamically backup user data files somewhat minimizes loss of the automatic recovery feature with this mode.

Transaction Commit Delay

c-treeACE supports grouping transaction commit operations for multiple clients into a single write to the transaction logs. This feature is referred to as a group commit or commit delay. Transaction commit delay is a good choice for optimizing performance in environments with large numbers of clients under high transaction rates. This is controlled with the `COMMIT_DELAY` keyword.

Commit delay helps decrease the overhead involved in flushing a transaction log. The performance improvement per individual thread may result in only milliseconds or even microseconds, however, multiplied by hundreds of threads and thousands of transactions per second, this amount becomes significant. FairCom has implemented a number of ways to enhance the effectiveness of commit delay logic.

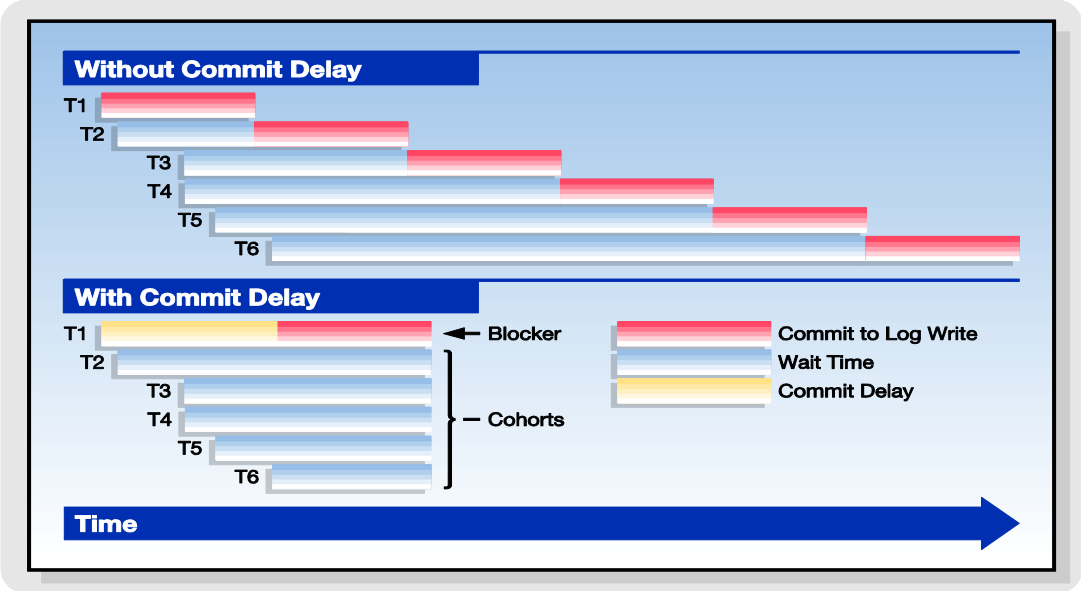
Commit Delay Operational Details

Without commit delay, each thread performs its own transaction log flush during a transaction commit. When commit delay is enabled, rather than each thread directly flushing the transaction log during a commit, threads enter the commit delay logic which behaves as follows.

In previous versions, any thread executing in the commit delay logic is known as either the blocker or a cohort. The blocker is the thread that eventually performs the transaction log flush on behalf of all threads waiting in the commit delay logic. A thread becomes the blocker on entry to the commit delay logic if there is not already a thread designated as the blocker. The blocker acquires a synchronization object (blocker), which is used to coordinate the threads (cohorts) that subsequently enter the commit delay logic. The blocker sleeps for the commit delay period specified in the server configuration file, wakes up, flushes the transaction log, and clears the block.

While the blocker is sleeping, other threads may enter the commit delay logic during their own transaction commit operations. These threads are known as the cohorts. The cohorts wait for the blocker to clear the block. When the blocker clears the block, each cohort acquires and releases the block, exits the commit delay logic without flushing the transaction log (because the blocker has already done this), and completes its commit operation.

Effect of commit delay on transaction commit times for multiple threads



The above figure shows the effect of commit delay on the commit times for individual threads. The left side of the figure shows the situation when commit delay is disabled. The right side shows the situation when commit delay is enabled. This example shows six threads (labeled T1 through T6) with random arrival times in the transaction commit log flush logic. In this example, the thread T1 arrives first, followed by thread T2 and so on through thread T6.

When commit delay is disabled, each thread flushes the transaction log in turn. The shaded part of the rectangles represents the time spent by each thread flushing the transaction log. Thread T1 flushes first. T2 waits until T1's flush completes and then performs its flush, and so on.

When commit delay is enabled, the first thread entering the commit delay logic becomes the blocker (thread T1 in this example). Threads entering the commit delay logic after this point in time (threads T2 through T6) become cohorts. The blocker sleeps for the commit delay period and then flushes the transaction log. The cohorts sleep until the blocker has finished flushing the transaction log and has released the block, at which point they acquire and release the block and complete their commit without flushing the transaction log.

An exception to the “blocker - cohort” concept arises when the log buffer becomes full prior to the delay period. In this instance, the cohort will flush thus releasing the blocker. Statistics are captured to measure this occurrence and to assess how the transactions flow through the commit delay logic.

Enabling Transaction Commit Delay

Commit delay can be enabled using either of these server configuration keywords:

```
COMMIT_DELAY <milliseconds>
```

where *<milliseconds>* is the commit delay interval specified in milliseconds, or:

```
COMMIT_DELAY_USEC <microseconds>
```

where *<microseconds>* is the commit delay interval specified in microseconds (one millisecond is 1000 microseconds).

If both forms of the commit delay keyword are used, then the last entry in the configuration file prevails.

Reduced Flushing of Updated Cache Pages

c-treeACE follows a buffer aging strategy, which ensures updated cache pages for transaction-controlled files are eventually flushed to disk. The factors that affect when a buffer is flushed include the number of times a buffer has been updated and the number of checkpoints that have occurred since the buffer was last flushed. This section describes ways to tune the c-treeACE buffer aging strategy to avoid unnecessary flushing of updated buffers for transaction-controlled files.

Transaction Flushing

The `TRANSACTION_FLUSH` configuration options controls the aging of updated buffers based on the number of times a buffer has been updated since it was last flushed.

`TRANSACTION_FLUSH <num_updates>` sets the maximum number of updates made to a data or index cache page before it is flushed. The default value is 500000. Increasing this value reduces repeated flushing of updated cache pages that may occur in a system that maintains a high transaction rate with a pattern involving frequently updating the same buffers.

Checkpoint Flushing

The `CHECKPOINT_FLUSH` server configuration keyword controls the aging of updated buffers based on the number of checkpoints that have occurred since the buffer was last flushed.

`CHECKPOINT_FLUSH <num_chkpnts>` sets the maximum number of checkpoints to be written before a data or index cache page holding an image for a transaction controlled file is flushed. Increasing this value avoids repeated flushing of updated cache pages that may occur in a system that maintains high transaction rates. When `CHECKPOINT_FLUSH` is increased, c-treeACE automatically detects the reliance on previous transaction logs and increases the active log count as needed provided the `FIXED_LOG_SIZE` configuration is not enabled.

The following formula estimates the number of logs required to support unwritten updated cache pages:

Let:

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

Then:

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

Note: c-treeACE does not use this formula. It dynamically adjusts for actual “recovery vulnerability” to determine exactly what logs are required for recovery.

Example

```
CPF=2,  CPL=3  => MNL = 3 (but the server enforces a minimum of 4)
CPF=19, CPL=3 => MNL = 9 active transaction logs
```

Improved Log Flush Strategy

The primary performance impact of transaction control is the result of flushing critical data to the transaction logs. Prior to V8.14, the c-treeACE Server flushed data to the transaction logs by issuing a write to the filesystem cache and then calling a system function to flush filesystem cache buffers to disk. FairCom found the most efficient way to flush data to the transaction logs is to open the logs using a synchronous write mode, in which writes bypass filesystem cache and write directly to disk.

c-treeACE Server versions 8.14 and later support transaction log write through using the following server configuration options for opening transaction logs in synchronous write mode:

- `COMPATIBILITY LOG_WRITETHRU` is used on Windows systems to instruct c-treeACE to open transaction logs in synchronous write mode. In this mode, writes to the transaction logs go directly to disk (or disk cache), avoiding filesystem cache, so the server avoid the overhead of first writing to the file system cache and then flushing the file system.
- `COMPATIBILITY SYNC_LOG` is used on Unix systems to instruct the c-tree c-treeACE to open its transaction logs in synchronous write (direct I/O on Solaris) mode. In this mode, writes to the transaction logs go directly to disk (or disk cache), avoiding the file system cache, so the server is able to avoid the overhead of first writing to the file system cache and then flushing the file system cache buffers to disk. This keyword also causes flushed writes for data and index files to use direct I/O. Using this keyword enhances performance of transaction log writes. (This option is deprecated as of V9 and replaced with `COMPATIBILITY LOG_WRITETHRU` for all platforms.)

Checkpoint Efficiency

Checkpoints are point-in-time snapshots of c-treeACE transaction states. c-treeACE writes checkpoints to the current transaction log at predetermined intervals. The most recent checkpoint entry placed in the transaction logs identifies a valid starting point for automatic recovery.

The interval at which checkpoints are written to the transaction log is determined by the server's `CHECKPOINT_INTERVAL` configuration keyword. This keyword specifies the number of bytes of data written to the transaction logs after which the server issues a checkpoint. It is ordinarily about one-third (1/3) the size of one of the active log files (*Lnnnnnnn.FCS*).

High transaction rate environments tend to generate large numbers of dirty cache pages which are flushed to disk during checkpoint processing. FairCom recommends increasing the checkpoint interval in these cases such that checkpoints occur less frequently reducing the volume of data flushed during the checkpoint. This can reduce the periodic latencies observed during checkpoint processing resulting in "flatter" overall I/O rates, and generally, overall better throughput.

Increasing the Interval Between Checkpoints

Two configurations interact in determining the minimum checkpoint interval.

1. Because the server enforces a minimum of three checkpoints per transaction log, increase the size of the transaction logs using the `LOG_SPACE` keyword. Set `LOG_SPACE` to 12 times the desired checkpoint interval to accommodate three checkpoints per log for four active transaction logs.

2. Set the `CHECKPOINT_INTERVAL` setting to the desired checkpoint interval.

For example, to set the checkpoint interval to 20 Mb, use a combination of these configuration options:

```
LOG_SPACE 240
CHECKPOINT_INTERVAL 20000000
```

Transaction Log Templates

Critical state information concerning ongoing transactions is saved on a continual basis in the transaction log file. A chronological series of transaction log files is maintained during c-treeACE operation. Transaction log files containing actual transaction information are saved as ordinary files and given names in sequential order, starting with *L0000001.FCS* (which can be thought of as “active c-treeACE Server log, number 0000001”) and incrementing sequentially (i.e., the next log file is *L0000002.FCS*, etc.). By default, c-treeACE retains up to four active logs at a given time.

By default, transaction log files are extended and flushed to ensure log space is available. Transaction logs are also 0xff filled to ensure known contents. The 0xff filling of the log file (and forcing its directory entries to disk) occurs **during** log write operations. For high transaction rate systems, this means log file processing is frequently busy with extension and fill processing, leading to increased latency for transactions in progress when log extension occurs.

Transaction log templates allows high throughput systems to maintain one or more preformed transaction logs ready to use. With log templates enabled, an empty log file is created at server startup, *L0000000.FCT*, to serve as a template. The first actual full log, *L0000001.FCS*, is copied from this template as well as the next blank log, *L0000002.FCT.* Whenever a new log is required, the corresponding blank log file is renamed from *L000000X.FCT* to *L000000X.FCS* and, asynchronously, the next blank log, *L000000Y.FCT*, is then copied from the template.

Log Template Configuration

Enable the log template feature by specifying the server keyword `LOG_TEMPLATE` in the server configuration file.

```
LOG_TEMPLATE <n>
```

where *<n>* is the number of log templates you want the server to maintain. The default is 0, which means no use of log templates. For instance, a value of two (2) means that two blank logs (*L0000002.FCT* and *L0000003.FCT*) would be created at first server startup in addition to the template (*L0000000.FCT*).

Prior to using the log template feature, existing transaction logs must be deleted to cause the server to create log templates. To do this, follow these steps:

1. Perform a **controlled shutdown** of the c-treeACE Server.
2. Block the ability of any clients to attach to the c-treeACE Server.
3. Perform a second **controlled shutdown** of the c-treeACE Server.
4. Remove all existing transaction logs and associated files: *Lnnnnnnn.FCS*, *S0000000.FCS*, and *S0000001.FCS*.
5. Unblock the ability of clients to attach to the c-treeACE Server and restart it.

When the server is restarted after adding this keyword, startup may take longer due to creation of template log files (**.FCT*).

Note: For more information, consult the procedures in the *Knowledgebase* section titled *Steps to Upgrade c-treeACE Server* (*http://docs.faircom.com/doc/knowledgebase/#product_upgradesteps.htm*) to find out how to cleanly shut down, delete the logs, and restart. Because you are not upgrading c-treeACE Server, you will ignore the step that tells you to copy the *ctsrvr.exe* or *ctreesql.exe* file.

Limitations

Log templates are not supported when mirrored logs or log encryption is in use.

Efficient Transaction Log Template Copies

The log template feature is a fast and efficient means of creating transaction logs in high volume systems. An initial transaction log template is created and copied when a new transaction log is required.

The original implementation used an operating system file copy command (for example., cp *L0000002.FCT L0000002.FCS*) to initiate the copy of the template to the newly named file. This approach required the full contents of the template file to be read. For systems experiencing high volume transaction loads where a template is frequently copied, this method placed unnecessary demand on system resources.

An improved efficient method for copying transaction log template file is available.

Template Copy Options

The following configuration options can be used to modify the speed of copying a log template such that log template disk write performance impact is reduced.

Copy Sleep Time

```
LOG_TEMPLATE_COPY_SLEEP_TIME <milliseconds>
```

This keyword results in the copying of the log template to be paused for the specified number of milliseconds each time it has written the percentage of data specified by the `LOG_TEMPLATE_COPY_SLEEP_PCT` option to the target transaction log file.

- Default value: 0 (disabled)
- Minimum value: 1
- Maximum value: 1000 (1 second sleep)

Copy Sleep Percentage

```
LOG_TEMPLATE_COPY_SLEEP_PCT <percent>
```

This keyword specifies the percentage of data that is written to the target transaction log file after which the copy operation sleeps for the number of milliseconds specified for the `LOG_TEMPLATE_COPY_SLEEP_TIME` option.

- Default value: 15
- Minimum value: 1
- Maximum value: 99

Example

The following example demonstrates the options that cause the copying of the log template file to sleep for 5 milliseconds after every 20% of the transaction log template file has been copied:

```
LOG_TEMPLATE_COPY_SLEEP_TIME 5
LOG_TEMPLATE_COPY_SLEEP_PCT 20
```

Note: If an error occurs using this method an error message is output to *CTSTATUS.FCS* (identified with the "LOG_TEMPLATE_COPY: ..." prefix) and c-treeACE then attempts the log template system copy method.

Efficient Flushing of Transaction Controlled Files

Similar to the strategy used in transaction log flushing (see `COMPATIBILITY LOG_WRITETHRU`), c-treeACE can flush transaction controlled data and index files with a file access mode bypassing filesystem cache. Two configuration options enable this behavior.

`COMPATIBILITY TDATA_WRITETHRU` and `COMPATIBILITY TINDEX_WRITETHRU` force transaction controlled data files and index files, respectively, to be written directly to disk (whenever c-tree determines they must be flushed from cache), and calls to flush their OS buffers are skipped.

Extended Transaction Number Support

Extended Transaction Number Support

The FairCom transaction processing logic used by the c-treeACE Server uses a system of transaction number high-water marks to maintain consistency between transaction controlled index files and the transaction log files. When log files are erased, the high-water marks maintained in the index headers permit the new log files to begin with transaction numbers which are consistent with the index files.

With previous releases, if the transaction number high-water marks exceed the 4-byte limit of 0x3ffffff0 (1,073,741,808), then the transaction numbers overflow, which will cause problems with the index files. On file open, an error **MTRN_ERR** (533) is returned if an index file's high-water mark exceeds this limit. If a new transaction causes the system's next transaction number to exceed this limit, the transaction fails with a **OTRN_ERR** (534).

6-byte transaction numbers essentially eliminate this shortcoming. With this new feature, 70,000,000,000,000 transactions can be performed before server restart. A transaction rate of 1,000 transactions per second would not exhaust the transaction numbers now available for over 2,000 years.

Note: The *Xtd8* file create and rebuild functions must be used in order to create files with 6-byte transaction number support. If a non-*Xtd8* file create function like **CreateIFileXtd()** is used to create index files, the index file is created without an extended header, so it cannot support 6-byte transaction numbers.

CreateIFileXtd8(), **PermIIndex8()**, **TempIIndexXtd8()**, and **RebuildIIfileXtd8()** are examples of functions that can be used to create indices with extended headers.

Even if the specified extended file mode does not include the 6-byte transaction number support flag (*ct6BTRAN*), c-tree defaults to using that option when a file is created with an extended header.

In c-tree Plus V8 and later, 6-byte transaction numbers are used by default. They will not be used on an individual file creation in the following cases:

1. If there is no extended create block; or
2. If the *ctNO6BTRAN* bit in the *x8mode* member of the extended file create block (*XCREblk*) is turned on; or
3. If the *ctNO_XHDRS* bit is turned on in *x8mode*.

You can override the default so that 4-byte transaction numbers are instead used by default by adding the `COMPATIBILITY 6BTRAN_NOT_DEFAULT` keyword to the server configuration file.

Ordinary data files are unaffected by this modification, and they are compatible back and forth between servers with and without 6-byte transaction support. Except for superfile hosts, the *ct6BTRAN* mode is ignored for data files. Index files and superfile hosts are sensitive to the *ct6BTRAN* mode: (1) an existing index file or superfile supporting only 4-byte transaction numbers must be converted or reconstructed to change to 6-byte transaction number support; and (2) the superfile host and all index members of a superfile must agree on their *ct6BTRAN* mode (either all must have the *ct6BTRAN* mode on or all must have it off), or a **S6BT_ERR** (742) occurs on index member creation.

Note: A previously existing index will only use 4-byte transaction numbers and an attempt to go past the (approx.) 1,000,000,000 transaction number limit will result in an **OTRN_ERR** (534). See Section 3.11.4 “Transaction High Water Marks” in the *c-treeACE Programmer’s Reference Guide* for more information.

Note: Files supporting 6-byte transactions are not required to be huge, but they do use an extended header.

An attempt to open a file using 6-byte transactions by code that does not support 6-byte transactions will result in **HDR8_ERR** (672) or **FVER_ERR** (43).

Configurable Extended Transaction Number Options

To check for files that do not support extended transaction numbers, add the following keyword to the c-treeACE Server configuration file:

```
DIAGNOSTICS EXTENDED_TRAN_NO
```

This keyword causes the server to log each physical open of a non-extended transaction number file to the *CTSTATUS.FCS* file. 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 exceed the original 4-byte range and one of these files is updated. By “all files” we mean superfile hosts and indices; data files are not affected by the extended transaction number attribute.

To enforce the use of only files with extended transaction numbers, add the following keyword to the c-treeACE Server configuration file:

```
COMPATIBILITY EXTENDED_TRAN_ONLY
```

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

These configuration options have no effect on access to non-transaction files, as transaction numbers are not relevant to non-transaction files.

Configurable Transaction Number Overflow Warning Limit

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

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

```
TRAN_OVERFLOW_THRESHOLD <transaction_number>
```

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

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

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

Efficient Single Savepoint for Large Transactions

The c-treeACE Server uses the **ReplaceSavePoint()** function internally to provide a fast, efficient means to carry along a save point in large transactions, so that an error can be undone by calling **RestoreSavePoint()** and then continuing the transaction. Compared to **SetSavePoint()**, which inserts a separate save point for each call, **ReplaceSavePoint()** simply updates some pre-image space links to effectively move the save point.

The **ReplaceSavePoint()** API call is now included in the c-tree client API. If your c-tree application needs the ability to undo only the last change in a transaction consider using **ReplaceSavePoint()**.

Deferred Flush of Transaction Begin

It is not uncommon for a higher-level application API to start transactions without knowledge of whether or not any updates will occur. To reduce the overhead of unnecessary log flushes, FairCom added a new transaction mode, *ctDEFERBEG*, to the c-tree API function **Begin()**, used to begin a transaction. *ctDEFERBEG* causes the actual transaction begin entry in the log to be delayed until an attempt is made to update a transaction-controlled file, and if a transaction commit or abort is called without any updates, then the transaction begin and end log entries are not flushed to disk.

FairCom applied this change after finding that c-treeACE SQL `SELECT` statements performed in auto-commit mode involved transaction log activity due to transaction begin and abort calls. c-treeACE SQL now includes this *ctDEFERBEG* mode in transaction begin calls, eliminating transaction log I/O for transactions that do not involve updates. If your application begins transactions that might not involve updates, consider adding *ctDEFERBEG* to your transaction begin calls.

Detection of Transaction Log Incompatibilities

c-treeACE transaction log formats periodically change as new features are added. Previously, error **LFRM_ERR** (666) indicated an existing transaction log was not compatible with the server or stand-alone application trying to read a log file at the start of execution.

A forward compatibility check, with description codes are now placed in *CTSTATUS.FCS* with additional details about log incompatibility. The forward compatibility check fails if an older build detects that transaction log requiring a feature not supported in the old logic.

Details about log incompatibility are demonstrated in this sample output:

```
Fri Sep 02 13:27:02 2005
  - User# 00001  Incompatible log file format...[7: 00400200x 00600290x]
```

The additional details are contained in the square brackets. The first number is an incompatibility code as described in the following table.

Incompatibility Code	Description
1	Log does not support *HUGE* files.
2	Log requires *HUGE* file support.
3	Log does not support 6-byte transaction numbers.
4	Log requires 6-byte transaction numbers.
5	Forward compatibility error: compare the two hex words.
6	Log does not support extended *TRANDSC* structure (*ctXTDLOG*).
7	Log requires extended *TRANDSC* support (*ctXTDLOG*).
8	Log does not support log compression (*ctCMPLOG*).
9	Log requires log compression support (*ctCMPLOG*).

The two hex numbers in the square brackets are bit maps. The first is derived from the log file and the second is from the build line. When the first number in the square brackets is 5, bits turned on in the first hex word, but not in the second hex word indicate which features the log requires that the logic does not support.

10. Configuring c-treeACE

Unless otherwise instructed, a c-treeACE Server starts using default settings for all configurable parameters. The c-treeACE Server takes configuration instructions from an encrypted configuration file, *ctsrvr.cfg*, placed in the c-treeACE Server directory. When the c-treeACE Server finds this file, it uses all the specified configuration values.

Note: Your vendor may also provide a settings file that is not user-configurable, *ctsrvr.set*.

Note: Keep in mind that the *ctsrvr.cfg* file needs to be saved in a Unix text file format. Failure to do so may cause an error to occur.

On Unix systems, keywords can be entered from the command line when starting the c-treeACE Server as described in *Advanced Configuration Keywords* (page 329, *http://docs.faircom.com/doc/ctserver/#65772.htm*).

Examples of reasons the c-treeACE Server may need to be reconfigured are:

- Communications protocols (for transmitting information to and from the c-treeACE Server): The default communications support for the c-treeACE Server is TCP/IP. Implementing other communication techniques requires a c-treeACE Server configuration file, and the appropriate `COMM_PROTOCOL` keyword.
- Memory allocations: To change the maximum amount of memory all users, or any given user, will be allocated—and to specify whether this maximum is an absolute rule or only a guideline.
- Backup files: To specify the c-treeACE Server should look for a dynamic dump script and follow instructions in that script to back up specified files.

10.1 c-treeACE Configuration File

To locally configure c-treeACE with a configuration file:

1. Create an ASCII file that is a list of configuration parameters using the file format, configuration keywords and values described in the *Configuration Options* (page 160) section.
2. Name this file *ctsrvr.cfg* and place in the appropriate directory. By default, c-treeACE looks in its executing directory for a file of this name when it starts.

   Tip: The default file name and path can be changed with an environment variable and a command-line keyword, as described in *Advanced Configuration Keywords* (page 329, *http://docs.faircom.com/doc/ctserver/#65772.htm*).
3. Start the c-treeACE Server.

Note: c-treeACE does not explicitly require a configuration file. If one is not present, c-treeACE starts with default values. Frequently, these values will not provide an optimal experience with the application.

Configuration File Format

The format of the c-treeACE Server configuration file is as follows:

- File name: *ctsrvr.cfg*
- Location of file: Same directory as c-treeACE Server executable
- File contents: An ASCII text file, consisting of a series of pairs of keyword names and keyword values, separated by one or more spaces (or a line feed). Keyword names are not case sensitive, but some values may be file names that may be case sensitive in certain environments, e.g., the value for `COMM_PROTOCOL`. Keyword values are strings of characters, without quotes around them, and without commas, decimal points, or indications of units (e.g., 1,000 bytes is entered as 1000). If a keyword is omitted from the configuration file, its default value is used.

For ease of reading and changing, we suggest using a format similar to the following configuration script file:

```
FILES        2000
CONNECTIONS  15
IDX_MEMORY   500000
DAT_MEMORY   500000
```

This configuration file specifies the following changes from default settings:

- `FILES`: Increase the maximum number of files from 1000 to 2000.
- `CONNECTIONS`: Set the maximum number of concurrent connections to the c-treeACE Server to 15.
- `IDX_MEMORY` and `DAT_MEMORY`: Increase the memory allocated to index cache and data cache, from 225,000 bytes to 500,000 bytes.

Individual lines in *ctsrvr.cfg* can be commented out with a semi-colon ';' in front of a line. Example:

```
;COMM_PROTOCOL F_TCPIP
```

Note: Multiple configuration files may be used. For example, create different configuration files for different dynamic dump schedules, or for different communication protocols. Be sure the appropriate configuration file is found by the c-treeACE Server when starting. Consider specifying the correct configuration via an environment variable, or passing on command line.

10.2 Configuration flexibility with environment variables

c-treeACE allows many of its configuration options to include an environment variable name that will be substituted with its value when the configuration file is read. For example, on Windows an administrator might want to set the c-treeACE data directory based on the `PROGRAMDATA` environment variable:

```
LOCAL_DIRECTORY %PROGRAMDATA%\FairCom\V10.1.0\bin\ace\sql\data
```

In this example, `%PROGRAMDATA%` is replaced with the value of the `PROGRAMDATA` environment variable (according to the environment variables that are defined in the logon session in which c-treeACE is running).

If you want to specify a literal `%` sign that is not treated as part of an environment variable name, use `%%`. For example if you want to specify a value of `ABC%DEF`, specify it as `ABC%%DEF`. A literal `%` sign cannot be used in an environment variable name; it always signifies the end of the environment variable name. For example `%MY%%VAR%` is treated as two environment variables, `MY` and `VAR`, rather than one variable named `MY%VAR`.

The following configuration options support this feature:

- `ADMIN_MIRROR`
- `BROADCAST_DATA`
- `CTSRVR_CFG`
- `DISK_FULL_ACTION`
- `DISK_FULL_VOLUME`
- `DYNAMIC_DUMP`
- `KEY_EXCHANGE_PARAMS`
- `LOCAL_DIRECTORY`
- `LOG_EVEN`
- `LOG_EVEN_MIRROR`
- `LOG_ODD`
- `LOG_ODD_MIRROR`
- `MASTER_KEY_FILE`
- `MIRROR_DIRECTORY`
- `NULL_STRING`
- `PREIMAGE_FILE`
- `SERVER_DIRECTORY` (deprecated)
- `SHMEM_DIRECTORY`
- `SIGNAL_DOWN`
- `SIGNAL_MIRROR_EVENT`
- `SIGNAL_READY`
- `SIGNAL_USER_DOWN`
- `SIGNAL_USER_READY`
- `SQL_LOGFILE`
- `START_EVEN`
- `START_EVEN_MIRROR`
- `START_ODD`
- `START_ODD_MIRROR`
- `TMPNAME_PATH`

10.3 c-treeACE Standard Wildcards

The c-treeACE standard wildcard pattern matching variables apply to the c-treeACE Server keywords that operate with files names, such as `MEMORY_FILES`, `KEEPOPEN_LIST`, `REPLICATE`, the `!FILES` command within the dynamic dump script, etc. The available list of wildcards is:

`*` - Multi-character match

`?` - Single-character match

`^` - Negation (must be first character)

For example, consider this list of files:

```
KEEPOPEN_LIST file1.dat
KEEPOPEN_LIST file2.dat
KEEPOPEN_LIST file30.dat
```

An asterisk can be used as a wildcard character to match `file1.dat`, `file2.dat`, and `file 30.dat` (or `fileanythingelse.dat`):

```
KEEPOPEN_LIST file*.dat
```

A `?` can be used to match a single character, which will match `file1.dat` and `file2.dat` but not `file30.dat`:

```
KEEPOPEN_LIST file?.dat
```

10.4 Scaling Factors for Configuration Keyword Values

The c-treeACE Server's data and index cache configuration options support specifying a scaling factor used when interpreting cache memory sizes. The supported scaling factors are:

- KB: interpret the specified value as a number of kilobytes.
- MB: interpret the specified value as a number of megabytes.
- GB: interpret the specified value as a number of gigabytes.

Example

```
DAT_MEMORY 100 MB
```

The following configuration keywords support scaling factors along with their limits.

Keyword	Limit
BUFR_MEMORY	1 GB
CHECKPOINT_INTERVAL	1 GB
CTSTATUS_SIZE	2 GB-1
DAT_MEMORY	2^32-1 cache pages on 64-bit systems and 4 GB-1 on 32-bit systems (although we allow up to 4 GB-1, the O/S might impose a lower limit; it will appear as c-tree error **10** when starting c-treeACE Server)
DISK_FULL_LIMIT	4 GB-1 for platforms without native 64-bit integer support;

	otherwise 2^63-1
DISK_FULL_VOLUME	4 GB-1 for platforms without native 64-bit integer support; otherwise 2^63-1
GUEST_MEMORY	2 GB-1
IDX_MEMORY	2^32-1 cache pages on 64-bit systems and 4 GB-1 on 32-bit systems (although we allow up to 4 GB-1, the O/S might impose a lower limit; it will appear as c-tree error **10** when starting c-treeACE Server)
LIST_MEMORY	1 MB
LMT_MEMORY	
LOG_PAGE_SIZE	32 KB
LOG_SPACE	1 GB
MAX_USER_LOG_ENTRY_BYTES	
MEMORY_FILE	2^63 - 1 bytes on 64-bit systems and 4 GB - 1 on 32-bit systems
MEMORY_MONITOR	
MEMORY_TRACK	
PAGE_SIZE	64 KB
PRIME_CACHE	2^63 - 1 bytes on 64-bit systems and 4 GB - 1 on 32-bit systems
PRIME_CACHE_BY_KEY	2^63 - 1 bytes on 64-bit systems and 4 GB - 1 on 32-bit systems
PRIME_INDEX	2^63 - 1 bytes on 64-bit systems and 4 GB - 1 on 32-bit systems
QUERY_MEMORY	
SORT_MEMORY	4 TB - 1 for 64-bit c-treeACE; 4 GB - 1 for 32-bit c-treeACE *(Prior to V10.3: 32 MB)*
SPECIAL_CACHE_FILE	2^32-1 cache pages on 64-bit systems and 4 GB-1 on 32-bit systems
TOT_MEMORY	
USR_MEMORY	2 GB-1

The following keyword was not changed to support specifying a scaling factor:

XTDKSEG_SRC_SIZE

Note: An operating system might impose a lower limit on the amount of memory available to a process than the configuration option limits that the c-treeACE Server supports

10.5 Alternative Configuration Methods

Settings File

Your vendor may supply an encrypted settings file, *ctsrvr.set*, that is not user configurable. Configuration options in this file are critical to the vendor's application, and should not be deleted. It is possible for the vendor to configure c-treeACE to require this file for startup. Review your vendor's documentation for guidance on this settings file.

Environment Variables

The c-treeACE Server supports environment variables specifying the location and file name of the Server Configuration, Settings, and License files, *FCSRVR_CFG*, *FCSRVR_SET*, and *FCSRVR_LIC* respectively. By default, c-treeACE looks for configuration, settings, and license files (named *ctsrvr.cfg*, *ctsrvr.set* settings file, and *ctsrvr*.lic*) in the configured working directory. This feature allows an administrator to specify alternate configuration for specialized purposes.

The environment variables, *FCSRVR_CFG*, *FCSRVR_SET*, and *FCSRVR_LIC* override the default names and locations when c-treeACE is launched. The environment variable should contain a complete file name for the configuration file.

For example, to direct c-treeACE to use *.\work\my_config.001* as its configuration file on a Windows platform, define *FCSRVR_CFG* as:

```
set FCSRVR_CFG=.\work\my_config.001
```

Likewise, *FCSRVR_SET* overrides the name and location of the Server Settings file:

(Unix example)

```
set FCSRVR_SET=/usr/production/my_set.abc
export FCSRVR_SET
```

FCSRVR_LIC overrides the default name and location of the c-treeACE Server license file. For example, to use *.\myFolder\ctsrvr12345678.lic* as the license file:

```
set FCSRVR_LIC=.\myFolder\ctsrvr12345678.lic
```

Command-Line Parameters

c-treeACE accepts configuration information from the command-line in addition to the settings and configuration files. Configuration keywords and values listed as command-line arguments take affect after an encrypted settings file, *ctsrvr.set*, if any, and before the standard configuration file, *ctsrvr.cfg*. A command-line entry cannot override a settings file entry, and a configuration file entry cannot override a command-line entry (or a settings entry). For more information on the encrypted settings file (*ctsrvr.set*), see additional documentation included in the **Security** section of the *Knowledgebase* in the *Customer Portal* (*https://support.faircom.com/*).

All valid configuration file keywords are supported and may be listed on the command line followed by an appropriate value. No special switch symbols or syntax is required. Simply enter each keyword followed by a value as follows:

```
ctreesql FUNCTION_MONITOR YES LOCAL_DIRECTORY C:\MYDATA\
```

or

```
ctsrvr FUNCTION_MONITOR YES LOCAL_DIRECTORY C:\MYDATA\
```

To specify the name and location of your server configuration file, *ctsrvr.cfg*, when launching the c-treeACE Server from the command-line, use the command-line keyword `CTSRVR_CFG` followed by a fully qualified configuration file name as follows:

```
ctreesql CTSRVR_CFG C:\myServer\ctsrvr.cfg
ctreesql CTSRVR_CFG  /usr/myserver/myinfo.cfg
```

or

```
ctsrvr CTSRVR_CFG C:\myServer\ctsrvr.cfg
ctsrvr CTSRVR_CFG  /usr/myserver/myinfo.cfg
```

The *CTSRVR_CFG* command line keyword is typically used when running two servers on the same machine, described elsewhere.

Note: The *FCSRVR_CFG* environment variable supersedes the `CTSRVR_CFG` keyword, so the file specified in the environment variable will always be the file used.

11. c-treeACE Configuration Options

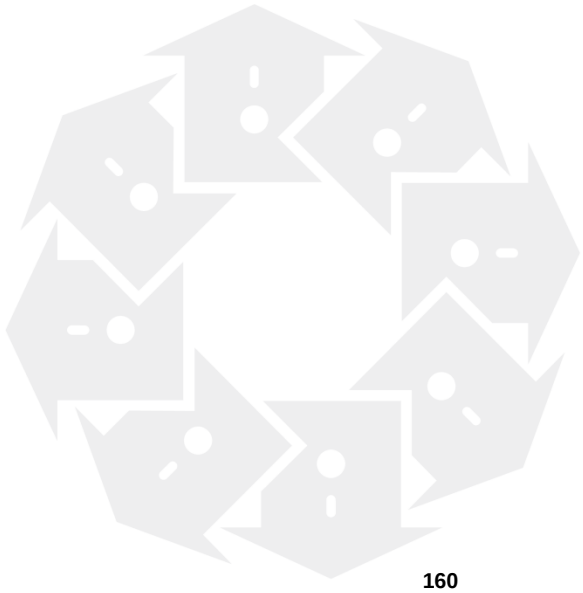
c-treeACE is extremely flexible and can be configured for nearly any environment.

Particularly common configurations include security controls, memory usage, performance options and transaction log management. File control options such as memory files, encryption and compression can also be centrally managed.

This flexibility comes with great responsibility. It is important end-user installations exactly follow application vendor recommendations for maintained performance and reliability. It is never advised to change production server settings without first confirming desired results in development settings. Catastrophic data loss can result from a mis-configured c-treeACE Server.

The keyword descriptions that follow use the nomenclature format in this table:

< input value >	An input value for the keyword. Replace with the value desired. For example, `COMMIT_DELAY <milliseconds>` would be put in the configuration file as `COMMIT_DELAY 10` to delay 10 milliseconds.
< value1 | value2 >	The | character indicates that either *value1* or *value2* can be used as described above, but not both together. For example, `ADMIN_ENCRYPT <YES | NO>` can be entered as either `ADMIN_ENCRYPT YES` or `ADMIN_ENCRYPT NO`.



11.1 Basic Keywords

The basic keywords are listed below. Each keyword links to a complete description along with legal and default values of each configuration option. The complete descriptions can be found in the section titled **Basic Options**.

Under normal circumstances, your configuration file should consist primarily of these keywords. The keywords listed in the other sections are restricted to more advanced configurations and should be used with caution.

COMMENTS (page 162)

Denotes that the remainder of the *ctsrvr.cfg* file is for documentation purposes only

COMM_PROTOCOL (page 162)

Specifies a communications module loaded by the server.

CONNECTIONS or USERS (page 164)

The maximum number of connections to the c-treeACE Server.

DAT_MEMORY (page 166)

The memory allocated to the data cache in bytes.

DUMP (page 166)

The name of a dynamic dump script file specifying when to begin and what to include in a dynamic dump.

FILES (page 167)

The maximum number of files to be open at one time.

GUEST_LOGON (page 169)

Controls whether or not to permit GUEST logons when no user ID is sent to the c-treeACE Server.

IDX_MEMORY (page 169)

The memory allocated to the index cache in bytes.

LOCAL_DIRECTORY (page 170)

Supplies the c-treeACE Server with the name of a directory path for processing files without absolute names.

MAX_DAT_KEY (page 170)

Maximum number of indices per data file.

MAX_KEY_SEG (page 171)

Maximum number of key segments allowed per index.

SERVER_NAME (page 171)

Assigns a name to the c-treeACE Server instead of the default name.

SERVER_PORT (page 172)

Specifies the TCP/IP Port of the server rather than using the `SERVER_NAME` keyword method.

COMMENTS

`COMMENTS`

Denotes that the remainder of the *ctsrvr.cfg* file is for documentation purposes only and is not to be used. The server ignores any remaining keywords. Comment individual lines by placing a semi-colon, ‘;’, at the beginning of the line.

Default: Off

COMM_PROTOCOL

```
COMM_PROTOCOL   F_TCPIP
COMM_PROTOCOL   F_TCPIPV6 (using an IPv6 socket)
COMM_PROTOCOL   FSHAREMM
COMM_PROTOCOL   DISABLE
```

Specifies a communications module loaded by the server. See *c-treeACE Server Installation* (page 7) for the communications options available for your platform.

If `COMM_PROTOCOL` is not specified, the default protocol is used.

The default protocol is disabled when the `COMM_PROTOCOL` keyword is used. If the default protocol is required in addition to other protocols, use the technique described below to specify multiple protocols.

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

IPv4 and IPv6

Native SQL clients on Windows will attempt to connect to the first address (IPv4 or IPv6) resolved for the host.

Note: In V11, IPv6 is supported on Windows and Linux. Java and ADO.NET SQL clients currently support only IPv4. Explicit IPv6 addresses in client connection strings are not currently supported.

Server keyword `SQL_OPTION NO_IPV6` can be added to *ctsrvr.cfg* to accept only IPv4 connections. This is not generally recommended and is more likely to cause connection issues than to solve them.

Note: The `COMM_PROTOCOL F_TCPIPV6` keyword is commented out in the default *ctsrvr.cfg* file. Be sure to remove the comment symbol (the semicolon - `;`) before trying to use this support.

An environment variable can be used for native clients to request only IPv4 address: *CTSQL_IPV4_ONLY*. Setting this variable to any value in the environment will effectively disable IPv6 connection attempts from the client. This may be needed on networks where both IPv4 and IPv6 are enabled, but the c-tree SQL server does not accept IPv6 connections.

Default Protocol Override

An application can override the default protocol by specifying the protocol in the connection string with the following syntax:

```
FAIRCOMS@myhost^TCPIP
```

or

```
FAIRCOMS@myhost^TCPIPV6
```

Shared Memory

On Windows, c-treeACE Server V10.3 and later can automatically detect and utilize shared memory when a TCP/IP client is in use. Include `COMM_PROTOCOL FSHAREMM` in the *ctsrvr.cfg* file for this to be triggered.

Multiple Protocols

c-treeACE Server on Windows, Linux, AIX, and Solaris (both SPARC and X86/X64) can support TCP/IP and shared memory simultaneously. For example, c-treeACE Server could be communicating with some users locally through shared memory and others using TCP/IP over an Ethernet connection.

Each protocol to be loaded by the c-treeACE Server requires a separate `COMM_PROTOCOL` line in the configuration script. The following example loads TCP/IP and Shared Memory Model protocols:

```
COMM_PROTOCOL  F_TCPIP
COMM_PROTOCOL  FSHAREMM
```

To support both IPv4 and IPv6 in the same c-treeACE Server, list the following keywords:

- `COMM_PROTOCOL F_TCPIP`
- `COMM_PROTOCOL F_TCPIPV6`

Note: If more than one protocol is to be used, a separate `COMM_PROTOCOL` entry must be specified for each protocol.

TCP/IP Encryption

Encryption of c-tree's TCP/IP communication protocol allows an added level of security for client/server applications. FairCom's proprietary encryption algorithm disguises communication packets between the client and the c-treeACE Server making it difficult for a casual user to inspect the information being exchanged. Since FairCom's proprietary encryption algorithm is designed primarily for performance, the *FETCPIP* protocol is only marginally slower (10-20%).

Use the server keyword `COMM_PROTOCOL FETCPIP` in the server configuration to specify encrypted TCP/IP communication. The c-treeACE Server simultaneously supports both encrypted and non-encrypted TCP/IP communication when both `COMM_PROTOCOL F_TCPIP` and `COMM_PROTOCOL FETCPIP` are specified in the server configuration. Clients must be compiled with either encrypted or unencrypted TCP/IP. The Application Developer will specify the appropriate protocol.

Identifying the c-treeACE Server Host Machine

Every protocol makes assumptions about the location of the machine hosting the c-treeACE Server. Use the following table to determine the proper method for the client to find the c-treeACE Server based on the protocol selected. `SERVER_NAME` is the name specified by the `SERVER_NAME` keyword, FAIRCOMS by default.

HostName is the network ID for the host machine. It can also be an IP address with TCP/IP.

Protocol	Default host	Specifying a host
Shared Memory	Local Machine	Only default can be used
TCP/IP	Localhost (127.0.0.1)	`SERVER_NAME`@HostName

Disable Communications

`COMM_PROTOCOL DISABLE` inactivates all communications.

When this keyword is specified in the c-treeACE Server configuration file, the server's communication subsystem is disabled at server startup, and remains disabled during the entire lifetime of the server process. This feature is useful when the c-treeACE Server is loaded as a DLL or shared library into an application server process. Although external clients are prevented from using the c-treeACE Server, threads in the application server process can use the c-treeACE Server subsystem. This option can also be used to prevent ISAM-level access to a c-treeACE SQL Server.

When this option is in effect, the server logs the following message to *CTSTATUS.FCS*:

```
c-treeACE Server communication subsystem is disabled.
```

Default: All servers load TCP/IP as the default.

CONNECTIONS or USERS

```
CONNECTIONS <Number of Connections>
```

The maximum number of connections to the c-treeACE Server. Typically, a c-treeACE Server is activated to support up to one of the following values for concurrent user connections: 8, 16, 32, 64, 128, 256, 512, or 1024. However, your particular c-treeACE Server may be customized with a

different value for connections. Specifying a number of users greater than the actual number of users needed results in inefficiencies (e.g., unused memory), so the goal is to keep this number as low as feasible on the system. The Activation Key flier displays the allowable number of users for your c-treeACE Server, as does the c-treeACE Server startup screen.

Default: Activated number of connections in the license

CPU_AFFINITY

CPU_AFFINITY <cpu list>

Windows

On Windows systems, `CPU_AFFINITY` server keyword can be used to set the processor affinity mask for the c-treeACE process. The option accepts a comma-delimited list of processor numbers. For example: `CPU_AFFINITY 0,1,2,3,8,9,10,11` indicates that c-treeACE is to be run on the eight specified CPUs.

If c-treeACE successfully sets the CPU affinity to the specified CPUs, the following message is logged to *CTSTATUS.FCS*, where *<cpulist>* is the list of CPUs specified for the `CPU_AFFINITY` option:

```
Successfully set CPU affinity to: <cpulist>
```

The following error situations can occur when using the `CPU_AFFINITY` option:

If the list of CPUs specifies a CPU number that is out of range on the system, a message is logged to *CTSTATUS.FCS*:

```
Configuration error: <config_file_name>, line <line_number>: The CPU_AFFINITY option specifies
an invalid CPU number for this system.
```

Solaris

On Solaris systems, `CPU_AFFINITY` accepts a single numeric value, which is interpreted as a processor set number. For example, `CPU_AFFINITY 2` configures c-treeACE to run on processor set 2.

> **Note**: To use `CPU_AFFINITY` under Solaris you need to run the c-treeACE Server with root permission. This because Solaris requires any process using a processor set to have such permission.

To create a processor set on Solaris, use the Solaris command `psrset`. For example, to create a set comprising processors 4 through 7, use:

```
psrset -c 4-7
```

where:

- *-c* - Create processor set.
- *4-7* - The processor numbers included in the set.

The ID of the newly created processor set is returned:

```
created processor set ps_id
```

To bind a process to this processor set, use:

```
psrset -b ps_id pid
```

where:

- *-b* - Bind.
- *ps_id* - The ID returned by the command when the processor set was created.
- *pid* - The ID of the process to be bound to the processor set.

If the process does not have permission to assign itself to the specified processor set (or if an invalid processor set is specified), c-treeACE logs the following message to *CTSTATUS.FCS*, where *<configuration_file_name>* is the name of the c-treeACE configuration file, *<line_number>* is the line number on which the `CPU_AFFINITY` option was specified, and *<error_code>* is the system error code returned by the OS.

Configuration error: <configuration_file_name>, line <line_number>: Failed to set CPU affinity: system error code <error_code>.

DAT_MEMORY

```
DAT_MEMORY <bytes>
```

The memory allocated to the data cache, specified in bytes. Within the memory constraints of the hardware, there is no effective limit.

Default: 100 MB

> **Note:** Prior to V11, the default value for both the standard c-treeACE Server and the c-treeACE SQL Server was 600 * `PAGE_SIZE`. Assuming a default page size of 8192, the default DAT_MEMORY would be 4915200.

See Also

`BUFR_MEMORY` (page 197)
`IDX_MEMORY` (page 169)
`TOT_MEMORY` (page 203)
`USR_MEMORY` (page 204)

DUMP

```
DUMP system.dmp
```

The name of a dynamic dump script file specifying when to begin and what to include in a dynamic dump. The contents of the script are described in "Dynamic Dump" (page 98). There is no default script, so the keyword `DUMP` does not appear in the default configuration file. An example configuration file entry for `DUMP` is:

> **Note:** If the `DUMP` keyword is used, the file named as containing the dynamic dump script must be in the same directory as the c-treeACE Server. The c-treeACE Server will look for this file only in its own directory and, if it does not find it, the c-treeACE Server will terminate immediately with error **FNOP_ERR** (12, file not found).

Default: None

See Also:

- PERMIT_NONTRAN_DUMP (page 279)
- DYNAMIC_DUMP_DEFER (page 278)
- DYNAMIC_DUMP_DEFER_INTERVAL (page 279)

FILES

```
FILES <Number of Files>
```

The maximum number of files to be open at one time.

There is no effective limit to the number of files supported by the c-treeACE Server, except for any limits imposed by the operating system and available system memory.

Note: The number of file descriptors set by the FILES keyword may need to be considerably greater than the number of open data files. Each index, whether or not in a separate file, counts toward this total. For example, a host index file that supports three different keys (i.e., contains three separate index members) counts as three files toward the FILES total. In addition, each member of a superfile is counted toward the total set by the FILES keyword.

Unix/Linux Considerations

The Unix and Linux operating systems place a limit on the number of file descriptors that can be in use at one time. The number of file descriptors required by c-treeACE is determined as follows:

- Each open file consumes 1 file descriptor (this may be somewhat lower than the setting of the FILES keyword because individual superfile members and members of a host index do not consume file descriptors).
- Each TCP/IP socket consumes a file descriptor (this corresponds to the CONNECTIONS keyword).
- c-treeACE reserves a few file descriptors in addition to those used for data files, index files, and TCP/IP connections.

Based on the above considerations, the system should be configured to allow a number somewhat greater than the number of **files + connections**.

The file descriptor limit is typically set by the **limit** or **ulimit** command, which may require superuser access, or by the hard limit set in a system configuration file such as */etc/security/limits.conf* on Linux. The specifics vary by system, so consult the documentation for your Unix or Linux system. (Unix and Linux define both hard limits and soft limits. A soft limit can be changed by a process at any time, but it cannot exceed the hard limit. The hard limit is of interest for this discussion.)

Note: When the file descriptor limit for c-treeACE Server (set by the operating system) is set too low, server startup fails with error **1005** (system-dependent initialization failed). A message is written to standard output indicating that system-dependent initialization failed and that details are in *CTSTATUS.FCS*.

New Features for Handling File Descriptor Limits

In V11 and later, several features improve the ability of the system to handle situations concerning file descriptors:

- *Improved file descriptor server startup messages* (page 168)
- *Fail server startup if file descriptor limit can't be increased to required value* (page 168)
- *Write message to standard output when file descriptor limit is too low* (page 169)
- *New file descriptor limit compatibility keyword* (page 169)

See Also

```
MAX_FILES_PER_USER (page 176)
COMPATIBILITY FILE_DESCRIPTOR_LIMIT (page 310)
```

Improved File Descriptor Limit Messages Logged During Server Startup

The file descriptor limit messages that c-treeACE Server logs to *CTSTATUS.FCS* at startup have been improved. The following information is logged:

- the file descriptor limit was successfully increased
- a warning that the maximum set by the system is not high enough to meet the server's file descriptor requirements
- the limit and the file descriptor requirement values

Now we also include the user limit in the file descriptor limit, because a TCP/IP socket uses a file descriptor.

Note: If the file descriptor limit cannot be increased to the required value, server startup fails. See *Fail server startup if file descriptor limit can't be increased to required value* (page 168).

Sample messages

Case #1: The system file descriptor limit is not high enough:

```
Mon Apr 28 13:17:55 2014
 - User# 00001  ERROR: The hard limit on file descriptors available to this process (4096) is lower
than the database engine's file descriptor requirement (11275). Either increase the hard limit,
or decrease the FILES or CONNECTIONS settings.
```

Case #2: c-treeACE was able to increase the limit:

```
Mon Apr 28 13:19:02 2014
 - User# 00001  Successfully increased current file descriptor limit to: 11275
```

Server Now Fails to Start if File Descriptor Limit Can't be Increased to Required Value

Appropriate operating system file descriptors are critical to c-treeACE server operation.

Note: If a file descriptor limit set by the operating system cannot be increased to the required value, server startup fails with error **1005** (system-dependent initialization failed).

If the file descriptor limit set by the operating system isn't high enough, it is possible to fail to open a transaction start file when performing a checkpoint, causing the server to terminate abnormally. This behavior avoids a runtime error by catching the insufficient file descriptor limit at server startup.

Message Written to Standard Output When File Descriptor Limit is too Low

When the file descriptor limit for c-treeACE Server (set by the operating system) is set too low, a message is written to standard output indicating that system-dependent initialization failed and that details are in *CTSTATUS.FCS*.

New file descriptor limit compatibility keyword

Although running c-treeACE Server with insufficient file descriptors can lead to errors opening files, we have added a compatibility keyword, `COMPATIBILITY FILE_DESCRIPTOR_LIMIT`, that restores the previous behavior in case it is not convenient for a system administrator to set the file descriptor limit for the c-treeACE Server process to the required value or it is not desired to decrease the FILES or CONNECTIONS settings.

> **Note**: Use of this keyword is generally discouraged. It is provided for backward compatibility or short-term use until site administrators are able to increase file appropriate file descriptor limits for a c-treeACE Server process.

A message is also logged to *CTSTATUS.FCS* explaining that the `COMPATIBILITY FILE_DESCRIPTOR_LIMIT` configuration option can be used to allow the server to start in this situation:

```
Tue Apr 29 12:23:44 2014
 - User# 00001  ERROR: The hard limit on file descriptors available to this process (500) is lower than the database engine's file descriptor requirement (1043). Either increase the hard limit, or decrease the FILES or CONNECTIONS settings.
Tue Apr 29 12:23:44 2014
 - User# 00001  Note: The configuration option COMPATIBILITY FILE_DESCRIPTOR_LIMIT can be used to allow c-tree Server to start even if the file descriptor limit is insufficient. However, this can lead to errors opening files.
```

GUEST_LOGON

```
GUEST_LOGON <YES | NO>
```

When no user ID is sent to the c-treeACE Server, the client is automatically assigned a user ID of GUEST. The keyword `GUEST_LOGON` controls whether or not to permit GUEST logons. The keyword takes YES or NO for its arguments.

Default:

Releases prior to V10: YES

Release V10.0 and later: NO

IDX_MEMORY

```
IDX_MEMORY <bytes>
```

The memory allocated to the index cache, specified in bytes. Within the memory constraints of the hardware, there is no effective limit. High-speed buffer search routines ensure quick access to the entire cache.

Default: 100 MB

> **Note:** Prior to V11, the default value for both the standard c-treeACE Server and the c-treeACE SQL Server was 600 * `PAGE_SIZE`. Assuming a default page size of 8192, the default `DAT_MEMORY` would be 4915200.

See Also

`BUFR_MEMORY` (page 197)
`DAT_MEMORY` (page 166)
`TOT_MEMORY` (page 203)
`USR_MEMORY` (page 204)

LOCAL_DIRECTORY

`LOCAL_DIRECTORY <Path>`

The preferred way to supply c-treeACE Server with the name of a directory path for processing files without absolute names. Absolute names include a specific volume or drive reference as part of the name, for example, *d:\fairserv\data*\. The trailing slash is required.

> **Note**: The other method, the `SERVER_DIRECTORY` configuration option, has been deprecated as of c-treeACE V9.3 and later. `LOCAL_DIRECTORY` is now the preferred keyword to allow the server to store data and files in an alternative location.

If a `LOCAL_DIRECTORY` name is defined in the configuration script, the name will be attached to the beginning of any file name that is not absolute. If neither `LOCAL_DIRECTORY` nor `SERVER_DIRECTORY` is supplied, database and system files are stored relative to the c-treeACE Server working directory. `LOCAL_DIRECTORY` and `SERVER_DIRECTORY` cannot be used together.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

> **Note:** The `LOCAL_DIRECTORY` does not become a permanent part of the file name. The name entered into the transaction log does not include the `LOCAL_DIRECTORY`.

Default: The directory where the c-treeACE Server process resides.

MAX_DAT_KEY

`MAX_DAT_KEY <Max Indices per Data File>`

Maximum number of indices per data file.

The typical error code that would be seen if this limit is too low is error 107, **IDRK_ERR** "Too many keys for ISAM data file."

> **Note**: In the standalone model, `MAX_DAT_KEY` is a compile-time setting, so the code must be recompiled if this setting is changed.

Default: 64 *(In V10.3 and earlier, the default was 32.)*

The default for the maximum number of indices per data file has been increased from 32 to 64.

Occasionally data files require a large number of indexes. Commencing with c-treeACE V10.3, the number of indices default limit was increased from 32 to 64. Customers using the c-treeACE Server can use the `MAX_DAT_KEY` keyword to change the limit on the number of indices per data file.

If this limit is too low, the typical error code that would be seen is error 107, **IDRK_ERR** "too many keys for ISAM data file."

This value affects the amount of memory that is allocated to store ISAM index state information. It can be increased without a major impact on performance unless c-treeACE Server is being run in an environment with very little memory (e.g., certain embedded applications).

Note: In the *standalone* model, `MAX_DAT_KEY` is a *compile-time* setting. If this value is changed, the code must be recompiled with the new setting.

MAX_KEY_SEG

```
MAX_KEY_SEG <Max Segments per Index>
```

Maximum number of key segments allowed per index.

Default: 16 *(Prior to V10.3, the default maximum number of segments per key was 12.)*

PAGE_SIZE

```
PAGE_SIZE <bytes>
```

The number of bytes per buffer page (maximum 65536 bytes). Only the following values are accepted (all other values generate an error):

- `1024`
- `2048`
- `4096`
- `8192`
- `16384`
- `32768`
- `65536`

To encourage compatibility across different c-treeACE environments, we suggest not modifying the `PAGE_SIZE`. However, if performance is of concern, this value may be modified to suit the characteristics of the operating system. Generally, this is a matter to discuss with the application programmer.

Default: 8192

SERVER_NAME

```
SERVER_NAME <NAME>
```

A name assigned to c-treeACE Server, instead of the default name.

Default: FAIRCOMS

See Also

- SERVER_PORT (page 172, http://www.faircom.com/doc/ctserver/48603.htm)
- COMPATIBILITY COMMPORT5000 (page 307)

SERVER_PORT

SERVER_PORT *<port_number>*

Specifies the TCP/IP Port of the server rather than using the SERVER_NAME keyword method. This allows for a direct specification of the port number.

With SERVER_NAME the TCP/IP port used is computed as 5001 plus the sum of the ASCII values of the characters in the server name. If both SERVER_NAME and SERVER_PORT are specified in the server configuration file, SERVER_PORT takes precedence over SERVER_NAME.

When a client prefixes the server name with the pound sign (#), the specified server name is interpreted as a numeric port. Otherwise, the specified server name is converted to a numeric port using the original approach.

For example: *#6000@localhost* is interpreted as port 6000, and *6000@localhost* is interpreted as port 5198.

Default

Off

See Also

- SERVER_NAME (page 171, http://www.faircom.com/doc/ctserver/27897.htm)
- COMPATIBILITY COMMPORT5000 (page 307)

11.2 Client Communication Keywords

c-treeACE clients generally connect to the server process via TCP/IP or shared memory. These options configure the parameters surrounding these connections.

c-treeACE also supports basic LDAP authentication and there are options for configuring those parameters.

DEAD_CLIENT_INTERVAL (page 175)

Controls the timeout interval (the number of seconds of client idle time after which the server checks the connection for that client).

MAX_CONCURRENT_USER_ACCOUNTS (page 175)

Sets the maximum number of user accounts that can connect to c-treeACE at one time.

MAX_CONNECTIONS_PER_USER_ACCOUNT (page 176)

Sets the maximum number of connections for each user account.

MAX_FILES_PER_USER (page 176)

Limits number of files per user when auto resizing comes into effect (e.g., a file operation uses a file number beyond the existing client file range or a new file number is assigned automatically).

MAX_ISAM_CONNECTIONS (page 177)

Sets the maximum number of ISAM connections.

MAX_SQL_CONNECTIONS (page 177)

Sets the maximum number of SQL connections.

Communications Protocol

COMM_PROTOCOL (page 162)

Specifies a communications module to be loaded by the server, which will determine the protocol to be used for client/server communications.

Each communication protocol has its own set of configuration keywords, listed below:

TCP/IP

BROADCAST_DATA (page 179)

Specifies a token to be broadcast following the Server Name.

BROADCAST_INTERVAL (page 179)

The number of seconds between broadcasts.

BROADCAST_PORT (page 179)

Specifies the TCP/IP port used for the broadcast.

SESSION_TIMEOUT (page 179)

Forces TCP/IP connections to be removed after the specified number of seconds has elapsed without activity.

Shared Memory

SEMAPHORE_BLK (page 181)

For Unix based systems only: the number of semaphores obtained at one time.

SHMEM_DIRECTORY (page 181)

Sets the directory in which c-treeACE stores files used for connecting using the Unix shared memory protocol.

SHMEM_PERMISSIONS (page 182)

Sets the permissions for the shared memory resources when using the Unix shared memory protocol.

LDAP

ADMIN_USER_GROUP (page 183)

Specifies the name of the super administrator user and administrator group.

GUEST_USER_GROUP (page 183)

Specifies the name of the guest user and guest group.

LDAP_BASE (page 184)

Specifies the base name for LDAP directory searches.

LDAP_SERVER (page 184)

Specifies the host name and port of an LDAP server for authentication.

LDAP_TIMEOUT (page 185)

Specifies an LDAP server connection timeout in seconds.

LOGIN_ALLOWED_GROUP (page 185)

If specified, only members of the specified group are allowed to connect to the c-treeACE Server.

Diagnostics

Diagnostics keywords are intended to be used to help collect data to be analyzed for troubleshooting. The use of these keywords should be avoided in normal operation as they may cause additional overhead (e.g., logging).

DIAGNOSTICS TRAP_COMM (page 186)

Instructs the c-treeACE Server to log incoming communications packets to *TRAPCOMM.FCS* prior to execution.

DEAD_CLIENT_INTERVAL

```
DEAD_CLIENT_INTERVAL idle_time_seconds
```

`DEAD_CLIENT_INTERVAL` controls the timeout interval, the number of seconds of client idle time after which the server checks the connection for that client. The default *nsec* value is 1800 (30 minutes), with a minimum of 120 (2 minutes).

`COMPATIBILITY TCPIP_CHECK_DEAD_CLIENTS` (page 327) is required to enable the checking. Otherwise, no check is made.

> **Note:** The timeout interval only controls how often the c-treeACE Server sends a message to test the connection. Different operating systems use different timeout values on TCP/IP messages, so the actual delay before a dead client is dropped will depend on when the operating system notifies the c-treeACE Server that the message failed.

Default: 1800 (30 minutes)

MAX_CONCURRENT_USER_ACCOUNTS

```
MAX_CONCURRENT_USER_ACCOUNTS <max_accounts>
```

Sets the maximum number of user accounts that can connect at one time to c-treeACE. The maximum number of ISAM and SQL connections can be set separately using `MAX_ISAM_CONNECTIONS` (page 177) and `MAX_SQL_CONNECTIONS` (page 177) respectively.

> **Note**: If limits are set in the license file, the configuration option can only be used to further reduce the connection limits (they cannot be increased above the license file limits).

The error code **ALMT_ERR** (984) is returned when logon is denied because the number of distinct user accounts that are allowed to be connected at one time has been reached.

See Also

- MAX_CONNECTIONS_PER_USER_ACCOUNT (page 176)
- MAX_ISAM_CONNECTIONS (page 177)
- MAX_SQL_CONNECTIONS (page 177)

MAX_CONNECTIONS_PER_USER_ACCOUNT

```
MAX_CONNECTIONS_PER_USER_ACCOUNT <max_connections>
```

Sets the maximum number of connections for each user account. The maximum number of ISAM and SQL connections can be set separately using MAX_ISAM_CONNECTIONS (page 177) and MAX_SQL_CONNECTIONS (page 177) respectively.

If a user is in a group, the connections are counted at the group level. If the user is not part of a group, connections are counted for the individual user.

Note: If limits are set in the license file, the configuration option can only be used to further reduce the connection limits (they cannot be increased above the license file limits).

The error code **ALMT_ERR** (984) is returned when logon is denied because the number of distinct user accounts that are allowed to be connected at one time has been reached.

See Also

- MAX_CONCURRENT_USER_ACCOUNTS (page 175)
- MAX_ISAM_CONNECTIONS (page 177)
- MAX_SQL_CONNECTIONS (page 177)

MAX_FILES_PER_USER

Client file information, on both client and server side, is automatically resized when:

- A file open or create uses a file number beyond the existing client file range.
- A new file number is assigned with automatic file number assignment (for example, **OpenFileWithResource()** with a -1 *filno*).

A configurable limit on files per user is enforced when this auto resizing comes into effect. The configuration keyword MAX_FILES_PER_USER defaults to 2048. An open/create which returns a **FINC_ERR** (604) implies that the create succeeded on the c-treeACE Server, but the client could not allocate memory for the local file info, and the newly created file has been closed.

Stand-alone applications support automatic resizing of file control information up to the limit imposed by *ctMAXFIL*, which defaults in *ctoptn.h* to 110. If *ctMAXFIL* is not defined, the limit defaults to 1024 files/FCBs.

Note: If a filno beyond the existing file range causes a resizing, the new number of files supported goes from 0 to filno + MAXMEMB + 1, with FCBs allocated for all potential file numbers in the range. Use automatic file number assignment for maximum memory efficiency.

For example, if **InitISAM()** requests 100 files and **OpenCtFile()** uses file number 1000, resizing changes the number of files supported to *1000 + MAXMEMB + 1*, or 1032. All the files between 100 and 1000 are now available. By contrast, if an automatic file number assignment causes resizing, the file number range is only extended by *MAXMEMB + 1*. If **InitISAM()** requests 100

files and **OpenFileWithResource(-1,...)** causes resizing, the number of files supported would increase to 132.

Default: 32767

See Also

FILES (page 167)
MAXMEMB (page 177)

Maximum Index Members per File (MAXMEMB)

In V11, the default for the MAXMEMB (the number of index members per single index file) has been updated to 127 allowing a larger number of segments per index.

We now define ctMAXMEMB in *ctopt1.h* instead of in *ctopt2.h*, because *ctport.h* references ctMAXMEMB and it is included before *ctopt2.h*.

We also define MAXMEMB to ctMAXMEMB in *ctopt1.h*.

MAX_ISAM_CONNECTIONS

MAX_ISAM_CONNECTIONS <max_number>

Sets the maximum number of ISAM connections. The maximum number of ISAM and SQL connections can be set separately using MAX_ISAM_CONNECTIONS (page 177) and MAX_SQL_CONNECTIONS (page 177) respectively.

Note: If limits are set in the license file, the configuration option can only be used to further reduce the connection limits (they cannot be increased above the license file limits).

The error code **ALMT_ERR** (984) is returned when logon is denied because the number of distinct user accounts that are allowed to be connected at one time has been reached.

See Also

- MAX_SQL_CONNECTIONS (page 177)
- MAX_CONCURRENT_USER_ACCOUNTS (page 175)
- MAX_CONNECTIONS_PER_USER_ACCOUNT (page 176)

MAX_SQL_CONNECTIONS

MAX_SQL_CONNECTIONS <max_number>

Sets the maximum number of SQL connections. The maximum number of ISAM and SQL connections can be set separately using MAX_ISAM_CONNECTIONS (page 177) and MAX_SQL_CONNECTIONS (page 177) respectively.

Note: If limits are set in the license file, the configuration option can only be used to further reduce the connection limits (they cannot be increased above the license file limits).

The error code **ALMT_ERR** (984) is returned when logon is denied because the number of distinct user accounts that are allowed to be connected at one time has been reached.

See Also

- MAX_ISAM_CONNECTIONS (page 177)
- MAX_CONCURRENT_USER_ACCOUNTS (page 175)
- MAX_CONNECTIONS_PER_USER_ACCOUNT (page 176)

TCP/IP

See COMM_PROTOCOL (page 162)

BROADCAST_DATA

```
BROADCAST_DATA <Token>
```

BROADCAST_DATA specifies a token to be broadcast following the Server Name. The token must not contain spaces. There is no default token. For example, add a department name or further identifying information for the c-treeACE Server.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: No data sent

See Also

BROADCAST_INTERVAL (page 179)
BROADCAST_PORT (page 179)

BROADCAST_INTERVAL

```
BROADCAST_INTERVAL <Seconds>
```

The number of seconds between broadcasts. The default is 10 seconds, otherwise the token should be a number. If the number is negative, each broadcast is also sent to the c-treeACE Server standard output.

Default: 10

See Also

BROADCAST_DATA (page 179)
BROADCAST_PORT (page 179)

BROADCAST_PORT

```
BROADCAST_PORT <DEFAULT | Port>
```

Specifies the TCP/IP port used for the broadcast. The default, 5595, is used when DEFAULT is specified, however, any valid four-byte integer greater than 5000 that is not in use by another process may be specified. This should NOT be the port for the c-treeACE Server, which is displayed at startup and is based on the Server Name. See the examples in "Advanced - Broadcast" (page 33).

Default: 5595

See Also

BROADCAST_DATA (page 179)
BROADCAST_INTERVAL (page 179)

SESSION_TIMEOUT

```
SESSION_TIMEOUT <seconds>
```

The SESSION_TIMEOUT option forces TCP/IP connections to be removed after the specified number of seconds has elapsed without activity. This option has been verified on Windows, Linux, and Mac OS X.

In V11 and later:

Improvements were made to the `SESSION_TIMEOUT` configuration option for 64-bit c-treeACE servers. Prior to these changes, an unhandled exception could occur in the administrative thread that checks for connections that have timed out due to inactivity. A timeout has been added to the socket receive so that each thread detects and performs its own disconnection in case of a timeout rather than using a separate administrative thread.

Additional changes were made to the processing of the `SESSION_TIMEOUT` configuration options:

- If `SESSION_TIMEOUT` is negative, it is ignored.
- If `SESSION_TIMEOUT` is less than 5, it is set to 5 so that the minimum SESSION_TIMEOUT value is 5 seconds.

Default: No timeout

Shared Memory

See COMM_PROTOCOL (page 162)

SEMAPHORE_BLK

```
SEMAPHORE_BLK <number>
```

Note: SEMAPHORE_BLK is a legacy keyword that is no longer used.

For Unix based systems only. This is the number of semaphores obtained at one time. These semaphores are used in the shared memory communication subsystem.

Default: 10

See Also:

- Shared Memory Client-Server Communication for Unix/Linux (page 22)

SHMEM_DIRECTORY

```
SHMEM_DIRECTORY <directory_name>
```

On Unix systems, the c-treeACE shared memory communication protocol creates a file that clients use to find the shared memory identifier for its shared memory logon region, and it creates named pipes for initial communication between client and server.

This option sets the directory in which c-treeACE stores files used for connecting using the Unix shared memory protocol.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

See Also

- SHMEM_PERMISSIONS (page 182)

SHMEM_GROUP

```
SHMEM_GROUP  <group>
```

By default, a client application must belong to the server owner's primary group to use shared memory. This is configurable with the SHMEM_GROUP keyword. This option causes c-treeACE Server to assign group membership to the specified group. This option applies to the resources for both the ISAM and the SQL shared memory protocol.

Possible errors indicating problems:

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

See Also

- System Group Assignment of Unix/Linux Shared Memory resources (page 30)

SHMEM_PERMISSIONS

`SHMEM_PERMISSIONS <permissions>`

On Unix systems, the c-treeACE shared memory communication protocol creates a file that clients use to find the shared memory identifier for its shared memory logon region, and it creates named pipes for initial communication between client and server.

`SHMEM_PERMISSIONS <permissions>` sets the permissions for the shared memory resources. The default is 660. 666 will allow access to c-treeACE by any user account.

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

See Also

- SHMEM_DIRECTORY (page 181)

LDAP

In V11 and later, c-treeACE Server's ability to check LDAP group membership has been improved. Previously, part of the filter was hard-coded. Now, the entire filter can be specified in the configuration file. Additionally, the attribute is no longer hard-coded; now it can be specified in the configuration file.

To use the new functionality, specify the following syntax for the `LDAP_ISAM_ALLOWED_GROUP` and/or `LDAP_SQL_ALLOWED_GROUP` options:

```
LDAP_ISAM_ALLOWED_GROUP {attr:ATTRIBUTE_VALUE}{base:BASE_VALUE}{filter:FILTER_VALUE}
```

For example:

```
LDAP_ISAM_ALLOWED_GROUP
{attr:member}{base:dc=mycompany,dc=com}{filter:(&(objectClass=groupOfNames)(cn=myusergroup))}
```

ADMIN_USER_GROUP

```
ADMIN_USER_GROUP <admin_user_name>:<admin_group_name>
```

Specifies the name of the super administrator user (default ADMIN) and administrator group (default ADMIN). Only members of the specified administrator group can perform some operations with c-treeACE such as shutting down c-treeACE or connecting using the **ctadmn** utility.

Example

```
ADMIN_USER_GROUP Administrator:Administrators
```

See Also

- LDAP_SERVER (page 184)
- LDAP_BASE (page 184)
- LDAP_TIMEOUT (page 185)
- GUEST_USER_GROUP (page 183)
- LOGIN_ALLOWED_GROUP (page 185)

Note: The LDAP authentication feature requires a custom build. Please contact your nearest FairCom office for current availability.

GUEST_USER_GROUP

```
GUEST_USER_GROUP <user:group>
```

Specifies the name of the guest user (default GUEST) and guest group (default GUEST).

Example

```
GUEST_USER_GROUP Guest:Guests
```

See Also

- GUEST_LOGON (page 169)
- LDAP_SERVER (page 184)
- LDAP_BASE (page 184)
- LDAP_TIMEOUT (page 185)
- ADMIN_USER_GROUP (page 183)
- LOGIN_ALLOWED_GROUP (page 185)

Note: The LDAP authentication feature requires a custom build. Please contact your nearest FairCom office for current availability.

LDAP_BASE

```
LDAP_BASE <ldap_base>
```

Specifies the base name for LDAP directory searches.

Example

```
LDAP_BASE DC=MyDomain,DC=local
```

See Also

- LDAP_SERVER (page 184)
- LDAP_TIMEOUT (page 185)
- ADMIN_USER_GROUP (page 183)
- GUEST_USER_GROUP (page 183)
- LOGIN_ALLOWED_GROUP (page 185)

Note: The LDAP authentication feature requires a custom build. Please contact your nearest FairCom office for current availability.

LDAP_SERVER

```
LDAP_SERVER  <ldap_host_name>:<ldap_port>
```

Specifies the host name and port of an LDAP server for authentication.

Example

```
LDAP_SERVER 192.168.0.15:389
```

See Also

- LDAP_TIMEOUT (page 185)
- LDAP_BASE (page 184)
- ADMIN_USER_GROUP (page 183)
- GUEST_USER_GROUP (page 183)

- LOGIN_ALLOWED_GROUP (page 185)

Note: The LDAP authentication feature requires a custom build. Please contact your nearest FairCom office for current availability.

LDAP_TIMEOUT

```
LDAP_TIMEOUT <timeout>
```

Specifies an LDAP server connection timeout in seconds (default is 60).

Example

```
LDAP_TIMEOUT 30
```

See Also

- LDAP_SERVER (page 184)
- LDAP_BASE (page 184)
- ADMIN_USER_GROUP (page 183)
- GUEST_USER_GROUP (page 183)
- LOGIN_ALLOWED_GROUP (page 185)

Note: The LDAP authentication feature requires a custom build. Please contact your nearest FairCom office for current availability.

LOGIN_ALLOWED_GROUP

```
LOGIN_ALLOWED_GROUP <group>
```

When this keyword is specified, only users who are members of the specified group are allowed to connect to the c-treeACE Server. The c-treeACE Server returns error **LGRP_ERR** when a user who is not a member of the specified group attempts to connect to the c-treeACE Server. If the keyword is not specified, any user who can be authenticated using the specified LDAP server is permitted to connect to the c-treeACE Server.

Example

```
LOGIN_ALLOWED_GROUP c-treeUsers
```

See Also

- LDAP_SERVER (page 184)
- LDAP_BASE (page 184)
- LDAP_TIMEOUT (page 185)
- ADMIN_USER_GROUP (page 183)
- GUEST_USER_GROUP (page 183)

Note: The LDAP authentication feature requires a custom build. Please contact your nearest FairCom office for current availability.

DIAGNOSTICS TRAP_COMM

`DIAGNOSTICS TRAP_COMM`

When activated, the `DIAGNOSTICS TRAP_COMM` keyword instructs the c-treeACE Server to log incoming communications packets to *TRAPCOMM.FCS* prior to execution. This log can be played back using the **cttrap** utility and a debug build of the c-treeACE Server to observe the results of the client requests, allowing the developer to exactly duplicate and repeat client activities. The trap file, *TRAPCOMM.FCS*, is created in the server directory by default. To prepend a path onto the trap file name (say to route it to a separate disk), add an entry of the form `DIAGNOSTIC_STR  <trap file path>`. For example, if `DIAGNOSTIC_STR /bigdisk/` were in the configuration file, then the trap file would be */bigdisk/TRAPCOMM.FCS*.

Notes

- A fresh *TRAMCOMM.FCS* file is created on each server startup and wipes out any existing one.
- *TRAPCOMM.FCS* isn't completely flushed until the server shuts down, thus it may appear empty until then.

Default: Disabled

See Also

`DIAGNOSTIC_STR (page 347)`
`DIAGNOSTIC_INT (page 347)`

11.3 Startup and Shutdown Keywords

APP_NAME_LIST (page 188)

Permits the server configuration file to create a list of arbitrary names that can be retrieved by clients.

CACHE_LINE (page 189)

Under multi-CPU systems, ensure this setting matches your equipment to increase performance by reducing false sharing in CPU caches.

CHECK_CONFIG (page 189)

Forces c-treeACE to continue processing the configuration file when an error occurs.

COMPATIBILITY NO_EXTERNAL_SHUTDOWN (page 189)

Disallow server shutdown attempts made by external c-tree clients.

CONSOLE CTRL_C_ENABLE (page 190)

Permit <CTRL>+C to stop the c-treeACE Server.

CONSOLE NO_MESSAGEBOX (page 190)

Deactivates error messages coming to the console in the form of a message box.

CONSOLE NO_PWRDWNPASSWORD (page 190)

Bypasses the ADMIN group User ID and password validation typically required at shutdown.

CONSOLE NO_SHUTDOWN_PROMPT (page 190)

Disables the shutdown prompt, which is used to acknowledge the shutdown to prevent an accidental unload.

CONSOLE TOOL_TRAY (page 190)

On Windows, starts the c-treeACE Server in background, displaying only an icon in the Windows tool tray.

CONSOLE W9X_SERVICE (page 191)

Windows 95/98 only: Execute the c-treeACE Server as a Windows 95/98 service.

CTSRVR_CFG (page 191)

Specify the configuration file used when executing the c-treeACE Server from the command line.

DNODEQ_SHUTDOWN_LIMIT (page 191)

Specifies a limit on the number of delete node queue entries that c-treeACE processes when it is shutting down.

NO_SHUTDOWN_FLUSH (page 192)

Skips the file flushing of the specified file during server shutdown.

PROCESS_PRIORITY (page 192)

Specifies the c-treeACE Server's process priority.

PROCESS_EXIT_COMMAND (page 192)

Runs a command when the c-treeACE process calls the system function **exit()**.

SIGNAL_DOWN (page 193)

Launches the specified application when the c-treeACE Server comes down.

SIGNAL_READY (page 193)

Launches an application when the c-treeACE Server comes up.

WAIT_ON_SHUTDOWN_SEC (page 193)

Permits explicit control on how long to wait for server shutdown.

Diagnostics

Diagnostics keywords are intended to be used to help collect data to be analyzed for troubleshooting. The use of these keywords should be avoided in normal operation as they may cause additional overhead (e.g., logging).

DIAGNOSTICS FULL_DUMP (page 193)

On Windows: Enables the generation of a mini-dump file with a *full* memory dump.

DIAGNOSTICS SHUTDOWN_COMM (page 194)

Logs messages showing the operations performed at server shutdown to disconnect client connections.

APP_NAME_LIST

`APP_NAME_LIST` permits the server configuration file to create a list of arbitrary names that can be retrieved by clients. The names are any text string not containing spaces. If spaces are

desired, the text string can be placed in quotes, however, the quotes will be part of the text string on retrieval.

As many APP_NAME_LIST entries as required may be created in the configuration file. For example: a list of file names may be created and then the names of the files can be retrieved by the client application. Further, by convention, one could use a delimiter to add parameters to the end of the string. Assuming, for the sake of example, that a pound sign (#) is used as a delimiter, then some entries could be added as follows:

```
APP_NAME_LIST       first.dat#parm11#parm12#parm13
APP_NAME_LIST       second.dat#parm21#parm22#parm23#parm24
```

It is up to the client application to determine how to parse the APP_NAME_LIST string that is returned in its entirety by calls to the c-tree **GetSymbolicNames()** API function.

The text string used in APP_NAME_LIST entries have the following restrictions:

1. The text string cannot contain a vertical bar (|) character; and
2. The string cannot exceed the maximum file name length (255 characters, or 512 characters when multi-byte characters are enabled).

CACHE_LINE

```
CACHE_LINE<size>
```

To maximize the performance of the c-treeACE Server under multi-CPU systems ensure the cache line setting matches the setting for your equipment. A cache-line is the smallest amount of memory a processor will retrieve and store in its highest speed cache. Using an appropriate CACHE_LINE setting helps reduce false sharing in CPU caches. Typical cache-line sizes are 32, 64, or 128 bytes.

It is common for 32-bit CPUs to use a 32-byte cache line size, and for 64-bit CPUs to use a 64-byte or larger cache line size, thus the default CACHE_LINE setting is 32 for 32-bit systems and 128 for 64-bit systems.

Default: 32 or 128

CHECK_CONFIG

```
CHECK_CONFIG  <YES | NO>
```

Forces c-treeACE to continue processing the configuration file when an error occurs. The server logs information about the success and failure of each keyword.

Note: CHECK_CONFIG currently only supports a minimal subset of configuration keywords. Errors with other keywords still terminate server processing of the configuration file.

COMPATIBILITY NO_EXTERNAL_SHUTDOWN

```
COMPATIBILITY NO_EXTERNAL_SHUTDOWN
```

c-treeACE can be configured to disallow server shutdown attempts made by external c-tree clients. (The term “external clients” refers to clients that communicate with the server through the server's communication subsystem.)

When this option is specified, attempts by external clients to shut down c-treeACE fail with error **XSHT_ERR** (792). This feature is most useful when c-treeACE is loaded as a DLL or shared library into an application server process. While external clients are prevented from shutting down the process, the application server process can shut down the c-treeACE subsystem by calling the **ctThrdTerm()** c-tree API function.

CONSOLE CTRL_C_ENABLE

`CONSOLE CTRL_C_ENABLE`

By default, c-treeACE Server ignores a <CTRL>+C signal. Adding this keyword to *ctsrvr.cfg* permits <CTRL>+C to stop the c-treeACE Server.

Default: Disabled

CONSOLE NO_MESSAGEBOX

`CONSOLE NO_MESSAGEBOX`

When activated for Windows machines, this keyword deactivates error messages coming to the console in the form of a message box. The c-treeACE Server continues to log messages to *CTSTATUS.FCS*.

Default: Disabled

CONSOLE NO_PWRDWNPASSWORD

`CONSOLE NO_PWRDWNPASSWORD`

Bypass the ADMIN group User ID and password validation typically required at shutdown. When this option is active and a machine shutdown or restart occurs, the prompt is bypassed and the c-treeACE Server shuts down cleanly.

Default: Request prompts

CONSOLE NO_SHUTDOWN_PROMPT

`CONSOLE NO_SHUTDOWN_PROMPT`

To prevent an accidental unload, the c-treeACE Server prompts the console to acknowledge the shutdown. The number of active connections is displayed and the user has the option to proceed with the shutdown (unload) or allow the server to continue running. Use the `CONSOLE NO_SHUTDOWN_PROMPT` keyword to disable this shutdown prompt.

Default: Show prompt

CONSOLE TOOL_TRAY

`CONSOLE TOOL_TRAY`

Under Microsoft Windows, starts the c-treeACE Server in background, displaying only a c-tree icon in the Windows tool tray. This feature can also be activated with the ampersand, '&', character on the command line. For example:

```
C:>ctreesql &
```

or

```
C:>ctsrvr &
```

In V11and later, the server waits until it has created the transaction logs, performed automatic recovery if necessary, and initialized the communication subsystem before displaying the balloon tip. This makes it easier for a developer or a server administrator to know the point in time at which c-tree Server is ready to accept connections.

Default: Disabled

CONSOLE W9X_SERVICE

It is possible to execute the c-treeACE Server as a Windows 95/98 service. Not to be confused with a Windows Server 2003/XP/Vista Service, this support is limited to the Windows 95/98 platform. This feature allows the server to remain in operation even if a user logs off of Windows without shutting down. Add the following keyword to the server configuration file to activate this support:

```
CONSOLE W9X_SERVICE
```

This keyword will be ignored on all other platforms except Windows 95/98.

Default: Disabled

CTSRVR_CFG

```
CTSRVR_CFG <path and filename>
```

Specify the configuration file used when executing the c-treeACE Server from the command line.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read. See *Environment Variables* (page 158) in *Configuring c-treeACE*.

Default: Server Working directory

DNODEQ_SHUTDOWN_LIMIT

```
DNODEQ_SHUTDOWN_LIMIT
```

c-treeACE supports a configurable limit on the number of delete node queue entries that c-treeACE processes when it is shutting down. If the option `DNODEQ_SHUTDOWN_LIMIT` is specified in the configuration file, then when c-treeACE shuts down, if there are more than the specified number of entries in the delete node queue, the delete node thread writes all the unique queue entries to a disk stream file named *DNODEQUE.FCS*. A memory-based index file is used to eliminate duplicate queue entries, and only the unique entries are written to disk. If the number of unique entries is less than `DNODEQ_SHUTDOWN_LIMIT`, those entries are returned to the delete node queue and are processed by the delete node thread before c-treeACE finishes shutting down.

`DNODEQ_SHUTDOWN_LIMIT 0` causes c-treeACE to process all entries in the delete node queue when shutting down.

c-treeACE always attempts to open and read all entries from the file *DNODEQUE.FCS* into the delete node queue at startup, regardless of the `DNODEQ_SHUTDOWN_LIMIT` setting. The

c-treeACE Server deletes the file *DNODEQUE.FCS* after populating the delete node queue with its contents.

An administrator can delete the file *DNODEQUE.FCS* before starting c-treeACE to avoid processing these persistent delete node queue entries.

NO_SHUTDOWN_FLUSH

```
NO_SHUTDOWN_FLUSH   <file name>
```

With very large (2GB+) caches, it may be possible for a file to never be written to disk during its entire life cycle. When the c-treeACE Server is shut down, it begins to flush files to disk. In the case of a “scratch” or “temp” file, the application vendor may not care if c-tree flushes this file to disk.

`NO_SHUTDOWN_FLUSH` skips file flushing during server shutdown. Note that *<file name>* may specify a wildcard pattern, with ‘?’ replacing a single character and ‘*’ replacing a group of characters. See **c-treeACE Standard Wildcards** (page 156). Non-transaction controlled files can be specified as shown below for this treatment, but such a file will be corrupted after shutdown if file flushing was actually skipped. Transaction-controlled files that are not flushed will simply lengthen automatic recovery times.

Default: Flush at shutdown

PROCESS_PRIORITY

```
PROCESS_PRIORITY
```

Specifies the c-treeACE Server's process priority.

Note: Use with caution, as a changing priority can substantially impact other processes on the system, or diminish database performance if decreased.

PROCESS_EXIT_COMMAND

```
PROCESS_EXIT_COMMAND <command>
```

Used to run a command when the c-treeACE process calls the system function **exit()**. Before running the command, c-treeACE sets the following environment variables:

CTREE_PROCESS_ID is set to the c-treeACE process ID

CTREE_SHUTDOWN_STATE is set to one of the following database engine shutdown status codes:

0 Shutdown not initiated.

1 Shutdown initiated .

2 Shutdown completed, but some clients remain active.

3 Shutdown completed with all files closed and final checkpoint logged.

SIGNAL_DOWN

```
SIGNAL_DOWN <executable>
```

Ability to launch an application when the c-treeACE Server comes down. This keyword takes as its argument the name of an executable that will be launched when the c-treeACE Server has been successfully terminated. Supported by c-treeACE Server for Unix and Windows.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: OFF

See Also

SIGNAL_READY (page 193)

SIGNAL_READY

```
SIGNAL_READY <executable>
```

Ability to launch an application when the c-treeACE Server comes up. This keyword takes as its argument the name of an executable, which will be launched when the c-treeACE Server is ready (i.e., automatic recovery is completed). Supported by c-treeACE Server for Unix and Windows.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: OFF

See Also

SIGNAL_DOWN (page 193)

WAIT_ON_SHUTDOWN_SEC

```
WAIT_ON_SHUTDOWN_SEC <seconds>
```

Permits explicit control on how long to wait for server shutdown. If all of the wait loops have been exhausted, and if it appears that some users are still not disconnected or that some files are still pending close, and if the wait specified with this key word has not been exceed, the server will retry its shut down loops. The new key word defaults to zero: no additional waiting. A negative values means to wait forever. A positive value specifies the time to wait. Due to sleeps within the shutdown loops, the actual wait time might exceed the specified time by an amount on the order of fifteen seconds, possibly more.

DIAGNOSTICS FULL_DUMP

```
DIAGNOSTICS FULL_DUMP
```

Enables the generation of a mini-dump file with a full memory dump, which are significantly larger in size.

Please note, if you have the `DIAGNOSTICS FULL_DUMP` keyword active, then any Windows Mini Dump Files (**.mdmp*) created will include a full memory dump of the c-tree Server process, which *may include end-user data*. If this keyword is not active, the dump contains only the c-tree Server stack trace, which does not include end-user data.

Note: This is only available on Windows.

DIAGNOSTICS SHUTDOWN_COMM

```
DIAGNOSTICS SHUTDOWN_COMM
```

Logs messages showing the operations performed at server shutdown to disconnect client connections.

Entries logged via this diagnostic option start with the text

```
shutdown_diag:
```

Additionally, with this option, the c-treeACE Server logs the details for each active thread that exists when the server begins to shut down, in the same format produced when using the `DIAGNOSTICS REMAINING_THREADS` keyword.

11.4 Cache and Memory Keywords

c-treeACE allows extremely fine tuned cache and memory management beyond basic data cache and index buffer pool sizes. This section covers options that control this granularity, including per user memory limits.

BUFBLK_RATIO (page 196)

Specifies the number of index buffer pages in each block list mutex.

BUFFER_RUNLENGTH (page 197)

Specifies the number of consecutive write operations performed while walking a list of buffer/cache pages before allowing other threads to acquire control of the list.

BUFR_MEMORY (page 197)

Specifies the size of memory blocks the c-treeACE Server uses in conjunction with data and index cache buffers.

COMPATIBILITY LARGE_CACHE (page 197)

Permits the `DAT_MEMORY` and `IDX_MEMORY` values to be reinterpreted as megabytes instead of bytes.

DATA_LRU_LISTS (page 198)

Specifies how many LRU lists are maintained for data cache pages.

GUEST_MEMORY (page 198)

Specifies the memory usage limit in bytes for a user without a User ID.

INDEX_LRU_LISTS (page 198)

Specifies how many LRU lists are maintained for index buffer pages.

LIST_MEMORY (page 199)

Specifies the size of memory “chunks” the c-treeACE Server uses for various internal data structures.

LMT_MEMORY (page 199)

Sets the maximum size for a single allocation using the c-treeACE memory suballocator.

MEMORY_HASH (page 199)

Causes 8 times the specified number of lists to be created (8 for each internal memory suballocator).

MPAGE_CACHE (page 200)

When set to a non-zero value, N, causes records falling within N+2 cache pages to be stored entirely within the cache.

NO_CACHE (page 200)

Specifies files that are NOT be cached.

PRIME_CACHE and PRIME_INDEX (page 201)

Specifies data and index files are added to the priming list and the number of bytes to be loaded into cache when physically opening the file.

PRIME_CACHE_BY_KEY (page 201)

Prime be the data cache from the specified file with the specified number of bytes by the specified index in forward or reverse order.

SORT_MEMORY (page 202, **http://docs.faircom.com/doc/ctserver/#27977.htm**)

Specifies the size of sort buffers used by the c-treeACE Server.

SPECIAL_CACHE_FILE (page 202)

Dedicates a specified amount of cache memory to a particular Extended data file.

TOT_MEMORY (page 203)

Specifies the total number of bytes the system will attempt to allocate for all uses (including index and data caches specified by the `IDX_MEMORY` and `DAT_MEMORY` keywords).

USR_MEM_RULE (page 204)

Sets the system default rule for c-treeACE Server action when a user exceeds his/her memory limit.

USR_MEMORY (page 204)

Specifies the system default limit of memory (in bytes) available to each user.

BUFBLK_RATIO

```
BUFBLK_RATIO <ratio of index buffer pages to block list mutexes>
```

The default ratio is 64. This means a block list mutex for every 64 index buffer pages. The `BUFBLK_RATIO` keyword can override the default. A ratio of one (1) means a mutex for every

buffer page. A configuration entry of zero is ignored (and the compile-time default is used). The advantage of the ratio specification is that the BUFBLK configuration entry does not have to be changed when the IDX_MEMORY entry is changed.

Note: Contact FairCom for details on this extremely advanced configuration option as this can severely impact online performance.

BUFFER_RUNLENGTH

```
BUFFER_RUNLENGTH <number of write operations>
```

This setting should be changed only at the request of your application developer. BUFFER_RUNLENGTH specifies the number of consecutive write operations performed while walking a list of buffer/cache pages before allowing other threads to acquire control of the list. A negative value is ignored.

Default: 10

BUFR_MEMORY

```
BUFR_MEMORY <bytes>
```

Specifies the size of memory blocks the c-treeACE Server uses in conjunction with data and index cache buffers. To minimize interaction with the underlying system memory manager, the c-treeACE Server manages its own blocks of memory out of which the buffer pages are allocated. The c-treeACE Server acquires one large block of memory and allocates smaller pieces as needed. If you are attempting to limit memory use by reducing IDX_MEMORY and/or DAT_MEMORY, set BUFR_MEMORY to about one eighth of the smaller of IDX_MEMORY and DAT_MEMORY.

Default: 64000

See Also

DAT_MEMORY (page 166)
IDX_MEMORY (page 169)
TOT_MEMORY (page 203)
USR_MEMORY (page 204)

COMPATIBILITY LARGE_CACHE

```
COMPATIBILITY LARGE_CACHE
```

To configure large data and index cache sizes, add COMPATIBILITY LARGE_CACHE to the c-treeACE Server configuration file. This keyword 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.

Example

```
COMPATIBILITY  LARGE_CACHE
```

```
IDX_MEMORY      100000000
DAT_MEMORY      4096
```

Requests 100 million bytes of index cache, and 4 GB of data cache.

Limitations

- The c-treeACE Server does not check that the specified amount of memory actually exists as available physical memory on the system. To avoid c-treeACE Server startup errors or performance degradation due to memory swapping, ensure that enough physical memory is available to accommodate the specified data and index cache sizes.
- The c-treeACE Server does not support this option on systems that do not support `LARGE_CACHE`. These systems are identified as 32-bit systems that do not have *SYS_LONG8* defined in their ***ctcmpl.h*** include file.

DATA_LRU_LISTS

```
DATA_LRU_LISTS <number of lists>
```

To reduce mutex contention, the cache control model permits the configuration keywords `DATA_LRU_LISTS` and `INDEX_LRU_LISTS` to specify how many LRU lists are maintained for data cache pages and index buffer pages, respectively.

In V10.3 and later on Windows, Solaris, and AIX, the default is based on the number of available CPUs on the system and the CPU limit specified in the license file. Otherwise, `DATA_LRU_LISTS` defaults to `4`.

See Also

`INDEX_LRU_LISTS (page 198)`

GUEST_MEMORY

```
GUEST_MEMORY <bytes>
```

If greater than zero, this is the memory usage limit in bytes for a user without a User ID (i.e., a GUEST user).

Default: 0

INDEX_LRU_LISTS

```
INDEX_LRU_LISTS <number of lists>
```

To reduce mutex contention, the new cache control model permits the configuration keywords `DATA_LRU_LISTS` and `INDEX_LRU_LISTS` to specify how many LRU lists are maintained for data cache pages and index buffer pages, respectively.

In V10.3 and later on Windows, Solaris, and AIX, the default is based on the number of available CPUs on the system and the CPU limit specified in the license file. Otherwise, `INDEX_LRU_LISTS` defaults to `4`.

See Also

`DATA_LRU_LISTS (page 198)`

LIST_MEMORY

```
LIST_MEMORY <bytes>
```

Specifies the size of memory “chunks” the c-treeACE Server uses for various internal data structures. To conserve memory, set this value to 4096. In situations with large amounts of available memory, the value can be increased beyond the default. The limit is 10 MB.

Default: 16384

LMT_MEMORY

```
LMT_MEMORY <limit>
```

Sets the maximum size for a single allocation using the c-treeACE memory suballocator. If this limit is exceeded a message is output to *CTSTATUS.FCS* and a NULL memory pointer is returned.

Default: 128 MB

> **Note:** A terr(7491) occurs,however, a informative message is sent to *CTSTATUS.FCS*.

In c-treeACE V11 and later:

Prior to this modification, c-treeACE Server's `LMT_MEMORY` configuration option defaulted to 128MB, meaning that a single memory allocation could not exceed 128MB. However, this limit restricted the size of variable-length records from c-treeACE Server and limited the amount of memory that could be allocated with the `RECOVER_MEMLOG` recovery option.

This limit has been *disabled* by default. If `LMT_MEMORY` is not specified, c-treeACE Server does not place a limit on the size of a single memory allocation. If desired, `LMT_MEMORY` can be specified in *ctsrvr.cfg* setting a desired maximum allocation size.

> **Note**: This modification results in a behavior change.

This change also applies to standalone mode: `#define ctMEMLMT` is set to zero by default, meaning no memory allocation size limit. If desired, c-treeACE can be compiled with *ctMEMLMT* defined to a non-zero value.

MEMORY_HASH

The c-treeACE internal memory suballocator utilizes 8 lists, each dedicated to a particular range of allocation size and each controlled by a single mutex. An expanded model improves scalability, especially when a large number of memory allocation or free operations take place at once.

The server configuration keyword `MEMORY_HASH <N>` causes 8N lists to be created, with N dedicated to a particular range of allocation size. In V10.3 and later on Windows, Solaris, and AIX, the default is based on the number of available CPUs on the system and the CPU limit specified in the license file. Otherwise, `MEMORY_HASH` defaults to `4`.

The `MEMORY_HASH` configuration requires aligned memory boundaries. If a particular server build is compiled without the proper alignment property, a messages is logged to CTSTATUS.FCS indicating this when this keyword is active:

```
- User# 00001      Configuration error: ctsrvr.cfg, line 3: This c-tree Server does not meet the
```

compile-time requirements to support the MEMORY_HASH keyword.

MPAGE_CACHE

MPAGE_CACHE <bytes>

The c-treeACE data cache uses the following approach to cache data record images:

- If the data record fits entirely within one or two cache pages (PAGE_SIZE bytes per cache page), then the entire record is stored in the cache.
- 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 efficient operations since they are aligned and are for a multiple of the cache page size.

The nature of this approach can be modified. Set MPAGE_CACHE to a value greater than zero, N, to store records falling within N+2 cache pages entirely within the cache. The default value is zero, behaves 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.

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

Default: 0

NO_CACHE

NO_CACHE <data file name>

In some cases, it might be beneficial to define that a certain file NOT be cached. For example, if a file contains very large variable length records (BLOBS), it might be more efficient to bypass the cache and rely solely on the operating systems cache support. The c-treeACE Server does not store the full variable length record in cache, but retains the first and last page of the variable length record. This prevents large blocks of data from consuming the cache and also alleviates the management of a large number of cache pages for any one particular record. NO_CACHE disables cache for a given file.

Note: *<data file name>* may include a wildcard pattern using '?' for a single character and '*' for zero or more characters. See **c-treeACE Standard Wildcards** (page 156). The Server Administrator can specify multiple NO_CACHE configuration entries.

Caching can only be turned off for entire superfiles (i.e., the superfile host), not for individual superfile members. Index files require the use of index buffer pages and must be cached.

Default: Cache files

PRIME_CACHE and PRIME_INDEX

The c-treeACE Server optionally maintains a list of data files and the number of bytes of data cache to be primed at file open. When priming cache, the server reads the specified number of bytes for the given data file into data cache when physically opening the data file.

Data files are added to the priming list with configuration entries of the form:

```
PRIME_CACHE   <data file name>#<bytes primed>
```

For example, the following keyword instructs the server to read the first 100,000 bytes of data records from *customer.dat* into the data cache at file open:

```
PRIME_CACHE   customer.dat#100000
```

A dedicated thread performs cache priming, permitting the file open call to return without waiting for the priming to be accomplished.

Use `PRIME_CACHE` with the `SPECIAL_CACHE_FILE` keyword to load dedicated cache pages at file open.

To prime index files, use configuration entries of the form:

```
PRIME_INDEX  <idx file name>#<bytes primed>[#<member no>]
```

If the optional *<member no>* is omitted, all index members are primed. If an index member number is specified, only that member is primed. For example, the following keyword instructs the server to read the first 100,000 bytes of index nodes in *customer.idx* into the index buffer space at file open:

```
PRIME_INDEX   customer.idx#100000
```

The nodes are read first for the host index, and then for each member until the entire index is primed or the specified number of bytes has been primed.

The following example restricts the priming to the first member (the index after the host index):

```
PRIME_INDEX   customer.idx#100000#1
```

The *<data file name>* or *<index file name>* can be a wildcard specification using a '?' for a single character and a '*' for zero or more characters. See **c-treeACE Standard Wildcards** (page 156).

Default: No priming

See Also

`PRIME_CACHE_BY_KEY (page 201)`

PRIME_CACHE_BY_KEY

```
PRIME_CACHE_BY_KEY <data_file_name>#<data_record_bytes_to_prime>#<index_number>
```

The `PRIME_CACHE` configuration option supports priming the data cache with the specified number of bytes of data from the specified data file, in physical order from the start of the data file. PRIME_CACHE_BY_KEY supports priming the data cache in forward AND reverse order by index.

- *data_file_name* specifies the name of the c-tree data file whose records are to be read into the c-treeACE Server's cache. The file name may include wildcard characters (see **c-treeACE Standard Wildcards** (page 156)).

- *data_record_bytes_to_prime* specifies the maximum number of bytes of data records to read into the c-treeACE Server's cache.
- *index_number* specifies the relative key number of the index to use when reading the data records. Specify 1 to indicate the first index, 2 to indicate the second index, etc., based on the index definitions stored in the data file's *IFIL* resource. Specify a negative value to indicate that records should be read in reverse key order by that index.

Example

```
PRIME_CACHE_BY_KEY mark.dat#100000000#-1
```

Primes up to 100,000,000 bytes from *mark.dat* reading by the first index in reverse key order.

See Also

PRIME_CACHE and PRIME_INDEX (page 201)

SORT_MEMORY

```
SORT_MEMORY <bytes>
```

Specifies the size of sort buffers used by the c-treeACE Server. To conserve memory, set this value to 8192 or 4096. If large amounts of memory are available, the value can be increased significantly beyond the default. This value must be less than the maximum segment size in segmented architectures.

The `SORT_MEMORY` keyword specifies the memory size in bytes and can use the MB and GB keywords (unlike `MAX_K_TO_USE`).

The maximum SORT_MEMORY value is:

- 4 TB - 1 for 64-bit c-treeACE
- 4 GB - 1 for 32-bit c-treeACE

As the `SORT_MEMORY` option is more intuitive, its use is recommended over MAX_K_TO_USE. (MAX_K_TO_USE remains available for backward compatibility). If both SORT_MEMORY and MAX_K_TO_USE are specified in *ctsrvr.cfg*, only the one that is specified *last* in the configuration file takes effect.

Default: 100 MB (16000 prior to V11)

See Also

MAX_K_TO_USE (page 335)

SPECIAL_CACHE_FILE

```
SPECIAL_CACHE_FILE  <datafilename>#<bytes to cache>
```

Dedicates a specified amount of cache memory to a particular Extended data file. This allows the Server Administrator to specify files that are critical to maintain in cache memory at the expense of the "least-recently-used" (LRU) approach, where a new cache page replaces the LRU existing page.

For example, the following keywords specify 100,000 bytes of dedicated cache for *customer.dat* and 400,000 bytes for *data\inventory.dat*:

```
SPECIAL_CACHE_FILE   customer.dat#100000
```

```
SPECIAL_CACHE_FILE   data\inventory.dat#400000
```

The *<datafilename>* can be a wildcard specification using a '?' for a single character and a '*' for zero or more characters. See **c-treeACE Standard Wildcards** (page 156).

Default: None

See Also

SPECIAL_CACHE_PERCENT (page 203)

SPECIAL_CACHE_PERCENT

```
SPECIAL_CACHE_PERCENT   <percentage>
```

Specifies the percentage of the overall data cache space that may be dedicated to individual files. For example, the following keyword would permit up to 10% of the total data cache pages to be assigned to files on the special cache file list:

```
SPECIAL_CACHE_PERCENT    10
```

To disable any special cache, enter -1 for the percentage. The percentage defaults to 50% and the maximum amount that can be specified with the keyword is 90%.

Default: 50%

See Also

SPECIAL_CACHE_FILE (page 202)

TOT_MEMORY

```
TOT_MEMORY <bytes>
```

If greater than zero, the total number of bytes the system will attempt to allocate for all uses (including index and data caches specified by the `IDX_MEMORY` and `DAT_MEMORY` keywords). If the system usage exceeds this level, attempts will be made to reduce discretionary allocations. If zero, no memory limit is imposed.

The `TOT_MEMORY` option will cause an operation to fail with an insufficient memory error such as **TSHD_ERR** (72) or **QMEM_ERR** (92) when the memory limit is reached. If this limit is exceeded, it may cause a user to flush preimage memory, but it will cause a **TSHD_ERR** (72) if the system limit is exceeded during preimage operations.

Memory management attempts to permit the server to survive exceeding the optional `TOT_MEMORY` limit. Although every effort is made to avoid this situation, it is possible that using this option could cause c-treeACE Server to terminate if it needs memory in a critical situation and cannot get memory, even if memory is available on the system. For these reasons, the use of this option can be risky.

Default: 0

See Also

BUFR_MEMORY (page 197)
DAT_MEMORY (page 166)
IDX_MEMORY (page 169)
USR_MEMORY (page 204)

USR_MEM_RULE

USR_MEM_RULE <GUIDELINE | ABSOLUTE>

The system default rule for c-treeACE Server action when a user exceeds his/her memory limit. This rule is employed only if the System Administrator has not assigned a rule specifically to the user or the user's primary group.

Valid values are:

- ABSOLUTE - The memory limit set for a user is to be applied as given, so no additional memory beyond the established limits will be allocated if it is requested.
- GUIDELINE - The memory limit set for a user guides memory allocation as follows: allow the user to have requested memory beyond the limit set, if it is available, and when another user needs that memory, then it reduces the amount of memory used back down toward the guideline as soon as possible (e.g., by moving memory resident information to disk).

Default: GUIDELINE

See Also

USR_MEMORY (page 204)

USR_MEMORY

USR_MEMORY

If greater than zero, this is the system default limit of memory (in bytes) available to each user. Zero means no user system default limit is imposed. The Administrator overrides this setting for a particular user by assigning a different value to the user or to the user's primary Group.

Default: 0

See Also

BUFR_MEMORY (page 197)
DAT_MEMORY (page 166)
IDX_MEMORY (page 169)
TOT_MEMORY (page 203)
USR_MEM_RULE (page 204)

11.5 Transaction Processing Keywords

Full ACID transaction processing of data is the gold standard for ensuring data integrity. Maintaining database changes in write-ahead logs to secure data is a complex and intricate process. As such, multiple features and options are available to fine tune this process for the best performance vs. integrity.

AUTO_PREIMG (page 209)

Allows newly created non-transaction files to be created in *PREIMG* mode.

AUTO_TRNLOG (page 209)

Extends the automatic transaction support to include recoverable transactions without any application change.

AUTO_TRNLOG_LIGHT (page 210)

Automatic transactions will be considered “light weight” so the transaction log is not flushed to disk on an automatic commit.

CHECKPOINT_FLUSH (page 210)

Sets the maximum number of checkpoints to be written before a buffer (data or index) holding an image for a transaction controlled file will be flushed.

CHECKPOINT_IDLE (page 211)

Specifies the time in seconds between checkpoint checks.

CHECKPOINT_INTERVAL (page 211)

Sets the interval between checkpoints (which can speed up recovery at the expense of performance during updates).

CHECKPOINT_PREVIOUS (page 211)

If the checkpoint used for automatic recovery does not appear to be valid, enables the server to attempt to start recovery at the location of the next to last checkpoint.

CHKPDFC_LOG_LIMIT (page 212)

Specifies how many consecutive logs may be processed without a checkpoint until the c-treeACE Server terminates.

COMMIT_DELAY (page 212)

Controls the time, in milliseconds, the transaction manager waits after a transaction completes before flushing the transaction to disk.

COMMIT_DELAY_BASE (page 213)

Determines the rate at which the nominal commit delay time is adjusted as the number of cohorts increases or decreases.

COMMIT_DELAY_SCALE (page 213)

For advanced control of the intricate commit delay timing statistics by adjusting the commit delay time calculation.

COMMIT_DELAY_USEC (page 213)

Same as `COMMIT_DELAY`, but interpreted as microseconds.

COMMIT_LOCK_DEFER_MS (page 214)

Provides an additional tuning mechanism for the `COMMIT_READ_LOCK` (page 317) retry value.

COMPATIBILITY LOG_WRITETHRU (page 214)

Instructs the c-treeACE Server to open its transaction logs in synchronous write mode.

COMPATIBILITY TDATA_WRITETHRU (page 215)

Forces transaction controlled data files to be written directly to disk, skipping the calls to flush their OS buffers.

COMPATIBILITY TINDEX_WRITETHRU (page 215)

Forces transaction controlled index files to be written directly to disk, skipping the calls to flush their OS buffers.

FIXED_LOG_SIZE (page 216)

Disables the c-treeACE Server from automatically adjusting the size of transaction log files to accommodate long records.

FORCE_LOGIDX (page 217)

Allows *LOGIDX* index file mode support to be forced on, off, or disabled.

KEEP_LOGS (page 217)

Specifies the number of non-active transaction log files kept on disk in addition to the active log files.

KEEP_RESTORE_POINTS (page 218)

Allows the server to maintain information about the last N Restore Points.

LOG_COMPRESSION_FACTOR (page 218)

Affects how compression is applied to log entries: if the compression does not fit in a buffer of size (<factor> * input_size / 100), then no compression takes place. Not currently supported.

LOG_COMPRESSION_THRESHOLD (page 218)

Affects how compression is applied to log entries: if the variable region is less than this threshold, no compression is attempted. Not currently supported.

LOG_EVEN (page 219)

The alternative name for even numbered transaction log files.

LOG_ODD (page 219)

The alternative name for odd numbered transaction log files.

LOG_PAGE_SIZE (page 219)

Changes the page size block of the transaction log buffer.

LOG_SPACE (page 219)

Specifies the number of megabytes of disk space allocated to storing active transaction logs.

LOG_TEMPLATE (page 220)

Enables transaction log templates.

LOG_TEMPLATE_COPY_SLEEP_PCT (page 220)

Specifies the percentage of data that is written to the target transaction log file after which the copy operation sleeps for the number of milliseconds specified for the `LOG_TEMPLATE_COPY_SLEEP_TIME` option.

LOG_TEMPLATE_COPY_SLEEP_TIME (page 221)

Causes the copying of the log template to pause for the specified number of milliseconds each time it has written the percentage of data specified by LOG_TEMPLATE_COPY_SLEEP_PCT to the target transaction log file.

LONG_TRANSACTION_MS (page 221)

Any transaction that exceeds the elapsed transaction time specified will be considered a long transaction for monitoring purposes.

MAX_USER_LOG_ENTRY_BYTES (page 221)

Specifies an optional limit for how many active transaction logs a transaction spans before it is aborted or abandoned.

MAX_USER_LOGS (page 222)

Controls how many logs a transaction can span before attempts are made to abort or abandon the transaction.

PREIMAGE_FILE (page 223)

The alternative name for the file containing preimages swapped to disk.

START_EVEN (page 223)

The alternative name for even numbered start file.

START_ODD (page 224)

The alternative name for odd numbered start file.

SUPPRESS_LOG_FLUSH (page 224)

Causes transaction begin and commit operations to skip the flushing of the log file when its argument is YES.

SUPPRESS_LOG_SYNC (page 225)

Skips the sync'ing to disk from a log flush operation.

TRAN_HIGH_MARK (page 225)

Specifies a transaction number threshold value.

TRAN_OVERFLOW_THRESHOLD (page 225)

Allows the server administrator to determine when the server will issue a warning that its transaction number is approaching the limit.

TRAN_TIMEOUT (page 226)

Sets a time limit on a transaction after which the transaction is aborted.

TRANSACTION_FLUSH (page 227)

Controls the maximum number of updates to a buffer (data or index) before it is flushed.

UNBUFFERED_LOG_IO (page 227)

Enables separate unbuffered I/O for transaction logs.

AUTO_PREIMG

```
AUTO_PREIMG <filespec>
```

Allows newly created non-transaction files to be created in *PREIMG* mode. These will automatically switch back to non-transaction when accessed by a non-transaction application. Files already created, but opened by a server with an applicable `AUTO_PREIMG` will be switched to *PREIMG* unless the file is an index without room for key level locks. Such a failed conversion at open is noted in *CTSTATUS.FCS*. *filespec* may contain wildcards.

Not only are files switched to *PREIMG* mode, but automatic ISAM *PREIMG* transactions will be launched. This permits an application to be run without any code change, and only a change to the configuration file if it is desired to use *PREIMG* files.

If a file is included in both `AUTO_PREIMG` and `AUTO_TRNLOG`, entries, then the `AUTO_PREIMG` entry will prevail.

Note: Consistency between data files cannot be ensured unless explicit transactions that preserve atomicity of updates are implemented by the application.

This also supports automatic transactions for low level operations if and only if the file in question has been opened under transaction control because of `AUTO_PREIMG` or `AUTO_TRNLOG`, configuration entries. By low level operations we mean updates to a data or index file that result from non-ISAM level API calls such as **WRTREC()**, **NEWVREC()** or **ADDKEY()**. Even if the file is included in one of these configuration entries, automatic low level transactions will not occur if the file is opened with inherent transaction attributes.

The server configuration keyword `PREIMAGE_DUMP` causes *PREIMG* files included in a dynamic dump to be automatically changed to *TRNLOG* files during the dump. Therefore, files converted by `AUTO_PREIMG` can be included in a dynamic dump that invokes the `PREIMAGE_DUMP` option. Each data file and associated indices will be restored to a consistent point.

Note: `AUTO_PREIMG` accepts wildcard file specifications (see **c-treeACE Standard Wildcards** (page 156)).

Files that already have transaction attributes are not affected by the configuration entries. An index file or c-tree Superfile host created without transaction support (and without these keywords) will not be able to be switched at open to transaction support. If such a file is included in the configuration lists for these key words, then the open will cause a *CTSTATUS.FCS* warning message with the name of the file.

See Also

```
AUTO_TRNLOG (page 209)
AUTO_TRNLOG_LIGHT (page 210)
DIAGNOSTICS AUTO_TRNLOG_CHECKLOCK (page 348)
DIAGNOSTICS AUTO_PREIMG_CHECKLOCK (page 348)
DIAGNOSTICS AUTO_TRNLOG_CHECKREAD (page 348)
DIAGNOSTICS AUTO_PREIMG_CHECKREAD (page 348)
PREIMAGE_DUMP (page 280)
```

AUTO_TRNLOG

```
AUTO_TRNLOG <filespec>
```

Extends the automatic transaction support to include recoverable transactions without any application change. AUTO_TRNLOG, is intended to permit transaction support for applications that do not make transaction calls. See the entry on AUTO_PREIMG for more details.

If a file is included in both AUTO_PREIMG and AUTO_TRNLOG, entries, then the AUTO_PREIMG entry will prevail.

Note: AUTO_TRNLOG accepts wildcard file specifications (see **c-treeACE Standard Wildcards** (page 156)).

Files that already have transaction attributes are not affected by the configuration entries. An index file or c-tree Superfile host created without transaction support (and without these keywords) will not be able to be switched at open to transaction support. If such a file is included in the configuration lists for these key words, then the open will cause a *CTSTATUS.FCS* warning message with the name of the file.

See Also

AUTO_PREIMG (page 209)
AUTO_TRNLOG_LIGHT (page 210)
DIAGNOSTICS AUTO_TRNLOG_CHECKLOCK (page 348)
DIAGNOSTICS AUTO_PREIMG_CHECKLOCK (page 348)
DIAGNOSTICS AUTO_TRNLOG_CHECKREAD (page 348)
DIAGNOSTICS AUTO_PREIMG_CHECKREAD (page 348)
PREIMAGE_DUMP (page 280)

AUTO_TRNLOG_LIGHT

```
AUTO_TRNLOG_LIGHT YES
```

AUTO_TRNLOG is intended to permit transaction support for applications that do not make transaction calls. In some instances, while the transaction support is desired, it may not be necessary to incur the full performance hit intrinsic with *TRNLOG* support. Adding AUTO_TRNLOG_LIGHT YES to a server configuration file means that the automatic transactions will be considered “light weight” which means that the transaction log entries are not required to be flushed to disk on an automatic commit. The default is NO: automatic transactions will invoke the strict transaction log management of regular transaction calls.

When AUTO_TRNLOG or AUTO_PREIMG are in effect, it may be desirable to know whether or not a legacy application is making “expected” lock calls

Default: **NO**

See Also

AUTO_PREIMG (page 209)
AUTO_TRNLOG (page 209)
DIAGNOSTICS AUTO_TRNLOG_CHECKLOCK (page 348)
DIAGNOSTICS AUTO_PREIMG_CHECKLOCK (page 348)
DIAGNOSTICS AUTO_TRNLOG_CHECKREAD (page 348)
DIAGNOSTICS AUTO_PREIMG_CHECKREAD (page 348)
PREIMAGE_DUMP (page 280)

CHECKPOINT_FLUSH

```
CHECKPOINT_FLUSH <# of checkpoints>
```

This keyword sets the maximum number of checkpoints to be written before a buffer (data or index) holding an image for a transaction controlled file will be flushed. The default value is 2. A value of zero causes the buffer to be flushed at least by the occurrence of the first checkpoint written after the buffer update. Reducing the value of this system parameter reduces the amount of buffering, slowing system performance, but decreases the amount of work to be performed during recovery.

Default: 17 (2 prior to V11)

See Also

`CHECKPOINT_INTERVAL (page 211)`

CHECKPOINT_IDLE

```
CHECKPOINT_IDLE <# of seconds | -1>
```

Specifies the time in seconds between checkpoint checks. A checkpoint is an entry in the transaction log which lists open files, active transactions and transactions that are vulnerable due to pending buffer flushes. By default, every 300 seconds the c-treeACE Server checks if there has been any transaction activity, and if so, if there are any current active transactions. If there has been activity since the last checkpoint, but there is currently no active transaction, a checkpoint occurs. This strategy will not create extra checkpoints when the c-treeACE Server is idle, with respect to transactions, or when the c-treeACE Server is busy with transactions.

It is important to note that if an application routinely calls **Begin()** whether or not updates are imminent, this “idle” checkpoint will be inhibited because there appears to be an active transaction. The purpose of this feature is to increase the likelihood of a clean checkpoint occurring in the transaction log, thus speeding automatic recovery. Ordinarily, checkpoints occur at predetermined intervals in the transaction log. A value of negative one (-1) will disable the idle checkpoint feature.

Default: 300

CHECKPOINT_INTERVAL

```
CHECKPOINT_INTERVAL  <interval in bytes or MB>
```

This keyword can speed up recovery at the expense of performance during updates. The interval between checkpoints is measured in bytes of log entries. It is ordinarily about one-third (1/3) the size of one of the active log files (*L000....FCS*). Reducing the interval speeds automatic recovery at the expense of performance during updates. The entry is interpreted as bytes if greater than 100 or as megabytes if less than 100. For example, `CHECKPOINT_INTERVAL 2` sets an approximate 2MB interval, while `CHECKPOINT_INTERVAL 150000` causes checkpoints about every 150,000 bytes of log file.

Default: 10 MB (833,333 bytes prior to V11)

See Also

`CHECKPOINT_FLUSH (page 210)`

CHECKPOINT_PREVIOUS

```
CHECKPOINT_PREVIOUS <YES | NO >
```

When automatic recovery begins, c-treeACE examines *S0000000.FCS* and *S0000001.FCS* to determine the location of the last checkpoint; which is the starting point for the automatic recovery. If the checkpoint does not appear to be valid (errors 64, **RLEN_ERR** and 66, **RCHK_ERR**), it is possible to have the server attempt to start the recovery at the location of the next to last checkpoint.

Default: NO

CHKPDFC_LOG_LIMIT

```
CHKPDFC_LOG_LIMIT  <max logs w/o checkpoint>
```

Ordinarily, several checkpoints are expected within each c-treeACE Server transaction log file and the c-treeACE Server would terminate with error **CHKP_ERR** (529) when two consecutive log files without these checkpoints were encountered. The c-treeACE Server has been modified with respect to this absence of checkpoints. This new behavior permits additional log files to be written without checkpoints. The key element of this change is that a short wait has been introduced for each non-checkpoint log write allowing a checkpoint to occur. This allows any possible race conditions to be resolved.

`CHKPDFC_LOG_LIMIT` specifies how many consecutive logs may be processed without a checkpoint until the c-treeACE Server terminates. The default number of transaction logs is now five log files, (previously only two logs were allowed) and this may be lowered to as few as four log files or raised to any reasonable limit.

Once the checkpoint deficiency reaches two logs, the first write of each transaction commit to the transaction logs is slightly deferred. This improve the chances that the c-treeACE Server checkpoint thread is allotted a slice of time. Once the checkpoint is eventually written, this write log defer is removed.

COMMIT_DELAY

```
COMMIT_DELAY <milliseconds | -1>
```

Controls the length of time in milliseconds after a given transaction completes that the transaction manager waits before flushing the transaction to disk. By waiting, more than one transaction (i.e., the first one and all others that complete before the delay period expires) may be committed at the same time reducing disk-access overhead. On average, the longer the delay, the larger the number of transactions committed.

Note: Keep this delay in mind when setting a time limit for aborting transactions.

Default:

Windows 2 ms.

Unix/Linux - in V11 and later, this value defaults to 1 ms for best performance.

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

See Also

- COMMIT_DELAY_BASE (page 213)
- COMMIT_DELAY_SCALE (page 213)
- COMMIT_DELAY_USEC (page 213)
- DELAYED_DURABILITY (page 216)

COMMIT_DELAY_BASE

```
COMMIT_DELAY_BASE   <cohort size measure>
```

For advanced control of the intricate commit delay timing statistics, additional controls are available for the commit delay time calculation.

The cohort size measure determines the rate at which the nominal commit delay time is adjusted as the number of cohorts increases or decreases. Increasing this value tends to reduce the amount of change in the blocking time as the number of cohorts changes.

It is recommended that these values be carefully profiled as they can impact performance in many unexpected ways.

Default: 50

See Also

```
COMMIT_DELAY (page 212)
COMMIT_DELAY_SCALE (page 213)
COMMIT_DELAY_USEC (page 213)
```

COMMIT_DELAY_SCALE

```
COMMIT_DELAY_SCALE  <ratio of block to clear time>
```

For advanced control of the intricate commit delay timing statistics, additional controls are available for the commit delay time calculation. The ratio of block to clear time cannot be smaller than 1. As this value increases, the amount of time waiting to permit commit delay cohorts to flow past their mutual block is decreased.

It is recommended that these values be carefully profiled as they can impact performance in many unexpected ways.

Default: 2

See Also

```
COMMIT_DELAY (page 212)
COMMIT_DELAY_BASE (page 213)
COMMIT_DELAY_USEC (page 213)
```

COMMIT_DELAY_USEC

```
COMMIT_DELAY_USEC <microseconds> | -1>
```

Same as `COMMIT_DELAY`, but interpreted as microseconds. If both forms of this keyword are specified, then the last entry in the configuration file prevails.

Note: Not all systems support arbitrarily short sleep times. FairCom has found, for example on the Solaris operating system, unless using real-time capabilities of the operating system, the minimum achievable sleep time is 10 milliseconds, even if a short sleep time is requested.

See Also

COMMIT_DELAY (page 212)
COMMIT_DELAY_BASE (page 213)
COMMIT_DELAY_SCALE (page 213)

COMMIT_LOCK_DEFER_MS

```
COMMIT_LOCK_DEFER_MS   <defer time in milliseconds>
```

Provides an additional tuning mechanism for the COMMIT_READ_LOCK (page 317) retry value.

The length of the defer value can be varied from zero to 100 milliseconds.

Internal tests demonstrated the affect of this change on CPU utilization was dramatic as a reader attempted to retry its read commit lock. Of course, actual performance increases will be variable, depending on any particular server environment. The trade-off with this method is introducing an unnecessary defer (i.e. if the next retry without a non-zero defer would have succeeded). In practice, this was not found to impede performance.

Commit write locks held by the transaction (i.e., locks that block read attempts during the actual commit process) are held during the entire commit. This has no direct impact upon the transaction commit, however, can cause longer delays for a read attempt when the transaction itself is comprised of a large number of write operations (e.g., committing thousands of **ADDREC()**s)

Default: 10

See Also

COMPATIBILITY NO_COMMIT_READ_LOCK (page 317)

COMPATIBILITY LOG_WRITETHRU

```
COMPATIBILITY LOG_WRITETHRU
```

Instructs the c-treeACE Server to open its transaction logs in synchronous write mode. In this mode, writes to the transaction logs go directly to disk (or disk cache), avoiding the file system cache, so the server is able to avoid the overhead of first writing to the file system cache and then flushing the file system. As of c-treeACE Version 9, this is applicable to both Windows and Unix Systems.

Note: On the Solaris operating systems, COMPATIBILITY LOG_WRITETHRU uses the O_DSYNC mode to implement synchronous log writes when available. (Direct I/O with O_SYNC is still used on those systems not supporting O_DSYNC.)

Default: OFF

Also See

- COMPATIBILITY DIRECT_IO (page 249)
- DELAYED_DURABILITY (page 216)

COMPATIBILITY TDATA_WRITETHRU

Similar to the strategy used in transaction log flushing, the c-treeACE Server can avoid excessive flushing of data and index files under transaction control. Two additional keywords affect this behavior:

`COMPATIBILITY TDATA_WRITETHRU` and `COMPATIBILITY TINDEX_WRITETHRU` force transaction controlled data files and index files, respectively, to be written directly to disk (whenever c-tree determines that they must be flushed from the c-tree buffers), and the calls to flush their OS buffers are skipped. These keywords cause transaction controlled files to be written through the OS file system cache, rather than written into the file system cache and later explicitly flushed at a database checkpoint. This behavior allows I/O costs to be evenly amortized, reducing the amount of I/O that must be done at a database checkpoint. This results in a smaller variance in transaction times, and potentially greater total transaction throughput. *For scenarios without heavy and continuous write activity, this alternate behavior frequently results in reduced throughput.*

`TDATA_WRITETHRU` uses the file system cache. The file is placed into a mode that causes the write to go to file system cache and then to disk before returning; the data still resides in file system cache. Compare to `UNBUFFERED_IO` (page 254), which completely avoids the file system cache.

See Also

`COMPATIBILITY TINDEX_WRITETHRU (page 215)`

COMPATIBILITY TINDEX_WRITETHRU

Similar to the strategy used in transaction log flushing, the c-treeACE Server can avoid excessive flushing of data and index files under transaction control. Two additional keywords affect this behavior:

`COMPATIBILITY TDATA_WRITETHRU` and `COMPATIBILITY TINDEX_WRITETHRU` force transaction controlled data files and index files, respectively, to be written directly to disk (whenever c-tree determines that they must be flushed from the c-tree buffers), and the calls to flush their OS buffers are skipped. These keywords cause transaction controlled files to be written through the OS file system cache, rather than written into the file system cache and later explicitly flushed at a database checkpoint. This behavior allows I/O costs to be evenly amortized, reducing the amount of I/O that must be done at a database checkpoint. This results in a smaller variance in transaction times, and potentially greater total transaction throughput. *For scenarios without heavy and continuous write activity, this alternate behavior frequently results in reduced throughput.*

See Also

`COMPATIBILITY TDATA_WRITETHRU (page 215)`

COMPATIBILITY LOCK_EXCL_TRAN

Skipping locks on exclusively opened TRNLOG files is enabled by default. It can be disabled by specifying the option `COMPATIBILITY LOCK_EXCL_TRAN` in *ctsrvr.cfg*.

DELAYED_DURABILITY

`DELAYED_DURABILITY <N>`

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

- When `DELAYED_DURABILITY` is set to 0 the new strategy is not in use.
- When `DELAYED_DURABILITY` is set to a positive value, <*N*>, the new strategy is in use and the log sync is guaranteed to occur within <*N*> seconds. A setting of 1 second is recommended because it results in a good performance gain (higher values offer very little additional benefit). The following configuration options are set as shown below:

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

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

Warning

When `DELAYED_DURABILITY` is enabled, recently committed transactions could be lost if c-treeACE Server terminates abnormally. For automatic recovery to succeed after c-treeACE Server terminates abnormally, either of the following must be done

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

   or

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

FIXED_LOG_SIZE

`FIXED_LOG_SIZE`

Long variable-length data records can cause a problem for transaction logs because they must be rolled over so fast checkpoints are not properly issued. By default, the c-treeACE Server automatically adjusts the size of the log files to accommodate long records. As a rule of thumb, if the record length exceeds about one sixth of the individual log size (2.5MB by default), the size is

proportionately increased. When this occurs, the *CTSTATUS.FCS* file receives a message with the approximate new aggregate disk space devoted to the log files.

YES in *ctsrvr.cfg* disables this feature. When disabled, ensure transactions do not last longer than is necessary. If a transaction is begun and is still open when the log size is exceeded, the server will terminate.

Default: NO

FORCE_LOGIDX

```
FORCE_LOGIDX <ON | OFF | NO>
```

`FORCE_LOGIDX` allows *LOGIDX* support to be forced on, off, or disabled:

- ON forces all indices to use *LOGIDX* entries.
- OFF forces all indices not to use *LOGIDX* entries.
- NO uses existing file modes to control *LOGIDX* entries.

The *LOGIDX* file mode is an index file mode permitting faster index automatic recovery during c-treeACE Server startup. Transaction controlled indices with this file mode are recovered more quickly than with the standard transaction processing file mode *TRNLOG*. This feature can significantly reduce recovery times for large indices and has not noticeably degraded the speed of index operations. *LOGIDX* is only applicable if the file mode also includes *TRNLOG*.

Note: The *LOGIDX* file mode is intended for index files only, and is ignored in data files. When adding the *LOGIDX* file mode to an existing index that is not under transaction control, be sure to rebuild the index to enable transaction control.

Default: ON

See Also

- DELAYED_DURABILITY (page 216)

KEEP_LOGS

```
KEEP_LOGS <number of inactive logs>
```

If greater than zero, `KEEP_LOGS` specifies the number of non-active transaction log files kept on disk in addition to the active log files. When a greater-than-zero `KEEP_LOGS` value is exceeded, the c-treeACE Server automatically deletes the oldest inactive log file as new log files are needed. If `KEEP_LOGS` is zero, inactive log files are immediately deleted by the c-treeACE Server. If `KEEP_LOGS` is -1, no inactive log files are deleted by the c-treeACE Server.

`KEEP_LOGS` permits the archiving of transaction logs. Inactive log files may be safely moved, deleted, copied or renamed. An inactive log file which is not immediately deleted by the c-treeACE Server is renamed from the form *L*.FCS* to the form *L*.FCA*. The last character in the extension is changed from ‘S’ to ‘A’, with the rest staying the same.

Default: 0

KEEP_RESTORE_POINTS

```
KEEP_RESTORE_POINTS <N>
```

`KEEP_RESTORE_POINTS <N>` allows the server to maintain information about the last *N* Restore Points. This is somewhat like the `KEEP_LOGS` keyword. The last *N* Restore Points are referred to as the "Active Restore Points." It is possible to set *N* to zero which means there will be no Active Restore Points. If there are no Active Restore Points, then automatic recovery cannot rollback to a quiet transaction state. The list of Active Restore Points is stored in each checkpoint. In the case of a Checkpoint Restore Point, the checkpoint includes itself as the last Active Restore Point.

Note: When *N* is greater than zero, the server automatically maintains the transactions logs necessary to ensure that a rollback to any of the Active Restore Points is possible. However, `KEEP_RESTORE_POINTS` does not affect the existence of the Restore Point files. These files are quite small (128 bytes), and are not deleted by the server.

See Also

- RECOVER_TO_RESTORE_POINT (page 230)

LOG_COMPRESSION_FACTOR

```
LOG_COMPRESSION_FACTOR  <percent>
```

Default: 80

See Also

- LOG_COMPRESSION_THRESHOLD (page 218)

Note: Log compression is not currently supported.

LOG_COMPRESSION_THRESHOLD

```
LOG_COMPRESSION_THRESHOLD  <bytes>
```

There are two parameters that control the manner in which the compression is applied to log entries.

- The first is a size threshold:
  If the variable region is less than this threshold, no compression is attempted.
- The second is a compression factor expressed as a percent:
  If the compression does not fit in a buffer of size (<factor> * input_size / 100), then no compression takes place.

Snapshot statistics include compression results.

Default: 100

See Also

- LOG_COMPRESSION_FACTOR (page 218)

Note: Log compression is not currently supported.

LOG_EVEN

```
LOG_EVEN <full_path>L
```

The alternative name for even numbered transaction log files. This name must be in the form of an optional directory path and the single character 'L' (e.g., *D:\LOG0\L*). The c-treeACE Server appends a seven-digit even number and the extension *.FCS* to the name given here.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

> **Note:** The ability to give separate device and directory names for odd and even log files allows them to be directed to different physical storage devices.

Default: L

See Also

LOG_ODD (page 219)

LOG_ODD

```
LOG_ODD <full_path>L
```

The alternative name for odd numbered transaction log files. This name must be in the form of an optional directory path and the single character 'L' (e.g., *D:\LOG1\L*). The c-treeACE Server appends a seven digit odd number and the extension *.FCS* to the name provided.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: L

See Also

LOG_EVEN (page 219)

LOG_PAGE_SIZE

```
LOG_PAGE_SIZE <bytes>
```

The c-treeACE Server uses a transaction log buffer to manage log write requests and this is comprised of a number of page size blocks. The current buffer size is 64K bytes. The LOG_PAGE_SIZE configuration key word is used to change the page size, and ideally, the log page size should match the optimal size for the disk I/O subsystem.

See Also

- PAGE_SIZE (page 171)

LOG_SPACE

```
LOG_SPACE <Megabytes>
```

This is the number of megabytes of disk space allocated to storing active transaction logs, starting with a minimum of 2. The c-treeACE Server maintains up to 4 active log files, which consume, in the aggregate, up to `LOG_SPACE` megabytes of disk space. Log files are numbered consecutively starting with 1. The log file names are in the form *L0000001.FCS*.

Default: 120 MB (10 prior to V11)

LOG_TEMPLATE

`LOG_TEMPLATE <n>`

Enables transaction log templates. *<n>* is the number of log templates for the server to maintain. A value of 0 means no use of log templates. A value of two (2) means that two blank logs (*L0000002.FCT* and *L0000003.FCT*) would be created at first server startup in addition to the template (*L0000000.FCT*). Log templates have been on by default since V9.

Before enabling log templates on a system that did not have them turned on, any existing transaction logs must be deleted so the server can create the log templates. To do this, shut down the server cleanly and delete *Lnnnnnnn.FCS*, *S0000000.FCS*, and *S0000001.FCS*. When the server is restarted after adding this keyword, initial startup may take longer due to creation of template log files (**.FCT*), however, using the templates will result in better performance in high transaction volume environments.

Default: 2

See Also

- LOG_TEMPLATE_COPY_SLEEP_TIME (page 221)
- LOG_TEMPLATE_COPY_SLEEP_PCT (page 220)

LOG_TEMPLATE_COPY_SLEEP_PCT

`LOG_TEMPLATE_COPY_SLEEP_PCT <percent>`

When working with large log templates on high volume systems, the log template copy operation can consume excessive IO time impacting performance. By periodically deferring the copy operation, this can smooth out IO spikes. This option, when used with the `LOG_TEMPLATE_COPY_SLEEP_TIME` option specifies the percentage of data that is written to the target transaction log file after which the copy operation sleeps for the number of milliseconds specified for the `LOG_TEMPLATE_COPY_SLEEP_TIME` option.

Default: 15
Minimum value: 1
Maximum value: 99

See Also

- LOG_TEMPLATES (page 220)
- LOG_TEMPLATE_COPY_SLEEP_TIME (page 221)

LOG_TEMPLATE_COPY_SLEEP_TIME

```
LOG_TEMPLATE_COPY_SLEEP_TIME <milliseconds>
```

When working with large log templates on high volume systems, the log template copy operation can consume excessive IO time impacting performance. By periodically deferring the copy operation, this can smooth out IO spikes. This option causes the copying of the log template to pause for the specified number of milliseconds each time it has written the percentage of data specified by the LOG_TEMPLATE_COPY_SLEEP_PCT option to the target transaction log file.

If an error occurs using this method to copy the log template, the code logs an error message to *CTSTATUS.FCS* (look for "LOG_TEMPLATE_COPY: ...") and attempts to use the original log template copy method.

A suggested time is 5 ms to start with.

Default: 0 (disabled)
Minimum value: 0
Maximum value: 1000 (1 second sleep)

See Also

- LOG_TEMPLATES (page 220)
- LOG_TEMPLATE_COPY_SLEEP_PCT (page 220)

LONG_TRANSACTION_MS

```
LONG_TRANSACTION_MS  <milliseconds>
```

A long transaction is any transaction that exceeds the elapsed transaction time specified. This will cause a message to be written to the *SYSMON* queue if the `PERF_MONITOR` configuration option is enabled.

See Also

- PERF_MONITOR (page 264)

MAX_USER_LOG_ENTRY_BYTES

```
MAX_USER_LOGS   <# of logs>
```

An optional limit for how many active transaction logs a transaction spans before it is aborted or abandoned. The default, ZERO, disables the check for long transactions.

When specified, `MAX_USER_LOGS` takes as its argument the maximum number of logs a transaction may span. If a transaction exceeds the limit, an attempt is made to abort the transaction. If the transaction cannot be aborted (consider the case where an abort would cause the server to fail), the transaction is abandoned. This means the client thread will lose its connection to the server, and the application may receive errors **ARQS_ERR** (127) or **ARSP_ERR** (128). There is no guarantee that a transaction will not span more logs than the specified maximum, however, the transaction will end within a reasonable number of logs.

If the transaction is aborted, then the next call by the client will return error **MLAB_ERR** (821) to indicate the operation was not completed and the pending transaction has been aborted. (See the

end note for a special case of this error condition.) A message of the following form will be made in *CTSTATUS.FCS*:

```
Sun Dec 03 08:53:21 2006
 - User# 00011  Transaction aborted at ct_mul_abandon1 for user# 9: 821
```

If the transaction is abandoned (that is, no explicit abort written in the log), then the client will be disconnected from the server. *CTSTATUS.FCS* entries such as the following reflect logs growing from a transaction that is pending, then the detection of the long transaction, then the eventual abandonment:

```
Sun Dec 03 09:53:42 2006
 - User# 00012  The number of active log files increased to: 5
Sun Dec 03 09:53:42 2006
 - User# 00012  Transaction (started in log 1) still pending.
                User# 11 |GUEST||
Sun Dec 03 09:53:55 2006
 - User# 00012  The number of active log files increased to: 6
Sun Dec 03 09:53:55 2006
 - User# 00012  Abandoned Transaction
Sun Dec 03 09:54:10 2006
 - User# 00012  The number of active log files increased to: 7
Sun Dec 03 09:54:10 2006
 - User# 00012  Abandoned Transaction2
Sun Dec 03 09:54:10 2006
 - User# 00012  Abandoned transaction kill request posted against user #11
        |GUEST||
Sun Dec 03 09:54:10 2006
 - User# 00011  ctntio: send error - O11 bytes=0 pErr=127
        |GUEST||: 168
Sun Dec 03 09:54:25 2006
 - User# 00012  The number of active log files decreased to: 4
```

The number of logs continued to grow, and then shrink, as reflected in the above excerpt because in addition to a transaction sleeping on a blocked lock, another unrelated application was continuing to add records to its files and corresponding entries in the transaction logs.

Note: In some rare situations error **TRAB_COD** (-823) can be returned instead of **MLAB_ERR**. This indicates the requested operation was completed before the abort actually took place. Usually, this is the same condition as an **MLAB_ERR**, as the transaction is aborted. In practice, the **TRAB_COD** should be rare.

Default: 0

MAX_USER_LOGS

```
MAX_USER_LOGS <max number of logs>
```

`MAX_USER_LOGS` controls how many logs a transaction can span before attempts are made to abort or abandon the transaction. The default, ZERO, disables the check for long transactions.

If a transaction cannot be aborted (for example, a server fault would occur) the transaction is abandoned which means that the client thread loses its connection to the server. There is no guarantee that the transaction will not span more logs than the specified maximum, but the transaction will end within a reasonable number of logs.

If the transaction is aborted, then the next call by the client will return a **MLAB_ERR** (821) to indicate the operation was not completed and the pending transaction has been aborted. A message of the following form will be made in *CTSTATUS.FCS*:

```
Sun Dec 03 08:53:21 2006
 - User# 00011  Transaction aborted at ct_mul_abandon1 for user# 9: 821
```

If the transaction is abandoned (which means no explicit abort written in the log), then the client will be disconnected from the server. *CTSTATUS.FCS* entries such as those shown below reflect logs growing from an transaction that is pending, then the detection of the long transaction, then the eventual abandonment:

```
Sun Dec 03 09:53:42 2006
 - User# 00012  The number of active log files increased to: 5
Sun Dec 03 09:53:42 2006
 - User# 00012  Transaction (started in log 1) still pending.
                User# 11 |GUEST||
Sun Dec 03 09:53:55 2006
 - User# 00012  The number of active log files increased to: 6
Sun Dec 03 09:53:55 2006
 - User# 00012  Abandoned Transaction
Sun Dec 03 09:54:10 2006
 - User# 00012  The number of active log files increased to: 7
Sun Dec 03 09:54:10 2006
 - User# 00012  Abandoned Transaction2
Sun Dec 03 09:54:10 2006
 - User# 00012  Abandoned transaction kill request posted against user #11
        |GUEST||
Sun Dec 03 09:54:10 2006
 - User# 00011  ctntio: send error - O11 bytes=0 pErr=127
        |GUEST||: 168
Sun Dec 03 09:54:25 2006
 - User# 00012  The number of active log files decreased to: 4
```

The number of logs continued to grow, and then shrink, as reflected in the above excerpt because in addition to a transaction sleeping on a blocked lock, another unrelated application was continuing to add records to its files and corresponding entries in the transaction logs.

PREIMAGE_FILE

```
PREIMAGE_FILE <Full_path>D
```

The alternative name for the file containing preimages swapped to disk. The format for this name is an optional directory path, which may include a Drive ID, followed by the single character ‘D’ (e.g., *E:\SWAP\D*). The c-treeACE Server appends a seven-digit number and the extension *.FCS* to the name provided here.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: D

START_EVEN

```
START_EVEN <full_path>S
```

The alternative name for even numbered start file. The start file contains the location at which the automatic recovery routines begin to scan the transaction logs. There are two start files (numbered zero and one) to reduce the risk of losing the starting point for automatic recovery. This name must be in the form of a directory path and the single character ‘S’ (e.g., *C:\START\S*). The c-treeACE Server appends a seven-digit even number and the extension *.FCS* to the name provided.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: S

See Also

START_EVEN_MIRROR (page 297)
START_ODD (page 224)
START_ODD_MIRROR (page 297)

START_ODD

START_ODD <full_path>S

The alternative name for odd numbered start file. The start file contains the location at which the automatic recovery routines begin to scan the transaction logs. There are two start files (numbered zero and one) to reduce the risk of losing the starting point for automatic recovery. This name must be in the form of a directory path and the single character 'S' (e.g., *C:\START\S*). The c-treeACE Server appends a seven digit odd number and the extension *.FCS* to the name provided.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: S

See Also

START_EVEN (page 223)
START_EVEN_MIRROR (page 297)
START_ODD_MIRROR (page 297)

SUPPRESS_LOG_FLUSH

SUPPRESS_LOG_FLUSH <YES | NO>

Causes transaction begin and commit operations to skip the flushing of the log file when its argument is YES. The default is NO. Suppressing the log flush makes it impossible to perform a proper automatic recovery. However, a dynamic dump will capture the necessary log information to restore *TRNLOG* files to a clean, consistent state. Using this keyword without the PREIMAGE_DUMP keyword is not recommended

By turning on PREIMAGE_DUMP and using *PREIMG* files, your system can run much faster than with full transaction processing, and still perform on-line dynamic dumps which will permit restoring files to the time of the dump in a clean, consistent state. However, it will NOT be possible to roll forward from the restored files because transaction log entries are not maintained outside of the dump process. See also PREIMAGE_DUMP and *Advanced - Faster Auto-Recovery* (page 98).

Note: We do not recommend disabling this keyword, as your data integrity will suffer. Be sure you understand what you are doing if you plan to change the default setting of this keyword.

Default: NO

See Also

- SUPPRESS_LOG_SYNC (page 225)
- DELAYED_DURABILITY (page 216)

SUPPRESS_LOG_SYNC

```
SUPPRESS_LOG_SYNC
```

Skips the sync'ing to disk from a log flush operation. This only applies to transaction begin/end operations. There are other system operations that cause log flushes, however, these are not affected. These other log flushes should also be relatively low in number.

Note: We do not recommend disabling this keyword, as your data integrity will suffer. Be sure you understand what you are doing if you plan to change the default setting of this keyword.

See Also

- SUPPRESS_LOG_FLUSH (page 224)
- DELAYED_DURABILITY (page 216)

TRAN_HIGH_MARK

```
TRAN_HIGH_MARK <long integer>
```

Specifies a transaction number threshold value. If a file is opened with a high-water mark value greater than this threshold, a message is placed in *CTSTATUS.FCS* listing the file name.

For example, the following configuration entry would cause any file whose high-water mark exceeds one million to have its name listed in *CTSTATUS.FCS*.

```
TRAN_HIGH_MARK  1000000
```

TRAN_OVERFLOW_THRESHOLD

```
TRAN_OVERFLOW_THRESHOLD <transaction_number>
```

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

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

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

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

operating). This keyword allows the server administrator to determine when the server will issue a warning that its transaction number is approaching the limit.

See Also

- Extended Transaction Number Support (page 149)

TRAN_TIMEOUT

`TRAN_TIMEOUT <interval>`

There are occasions where it is valuable to limit the time that a c-treeACE Server transaction is allowed to span. Long held transactions can cause a number of application-related issues. For example, holding locks on a record, or preventing updates to be available to other users in a timely manner.

`TRAN_TIMEOUT` sets a time limit on a transaction: when a transaction is started, once the time period passes, if the transaction is still active it is aborted regardless of what the user is doing (the user could be idle for example).

- `<interval>` is specified in seconds.
- The minimum value for the timeout `<interval>` is 10 seconds. Any value between 1 and 10 is the same as 10.
- If the `<interval>` is set to 0 (or a negative number), this feature is turned off.

`TRAN_TIMEOUT` is also useful to avoid increases in the number of active transaction logs which can occur due to a user starting a transaction and then remaining idle without committing the transaction, while other transaction activity occurs. `TRAN_TIMEOUT` aborting the transaction releases locks acquired within the transaction.

Deferred Transaction Begins

A deferred begin transaction (a transaction started using *ctDEFERBEG*), only starts counting the transaction time when the transaction is converted to an actual transaction (typically on the first update made in that transaction). For example, if you start **isql** and do some SELECTS then look at the transaction time shown for the SQL connection by **ctadmn**, it will show `'--'` indicating the transaction is not yet an actual transaction (this is because SQL threads use *ctDEFERBEG* transactions). Once you perform an update, **ctadmn** will show the transaction time counting, and if `TRAN_TIMEOUT` is in effect, the transaction will be aborted if it does not commit before the `TRAN_TIMEOUT` limit.

Transaction Timeout Statistics

The **USERINFO()** function returns state information for a particular connection to the c-treeACE Server. Included in the state information is the elapsed transaction time for that connection. When a transaction is started with the *ctDEFERBEG* mode, the elapsed transaction time value returned by **USERINFO()** is based on the time at which the **TRANBEG()** call was made.

For a transaction whose begin has been deferred, **USERINFO()** returns an elapsed transaction time of zero until the transaction begin is converted to an actual transaction begin, at which point the elapsed transaction time is calculated from that time.

Default: No timeout

TRANSACTION_FLUSH

```
TRANSACTION_FLUSH <# of updates>
```

This keywords provides control for the maximum number of updates to a buffer (data or index) before it is flushed. The buffer may well be flushed prior to this number of updates because of the LRU (Least Recently Used) scheme or because of the checkpoint limit. This system parameter affects only buffers holding images for transaction controlled files. Reducing this value reduces the amount of buffering, slowing system performance; but decreases the amount of work to be performed during recovery. A value of zero causes the buffer to be flushed upon update.

Default: 500000

UNBUFFERED_LOG_IO

```
UNBUFFERED_LOG_IO <YES | NO>
```

Enable separate unbuffered I/O for transaction logs.

OS Support

This option is supported on the Windows operating system.

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

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

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

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

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

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

See Also

- UNBUFFERED_IO (page 254)

11.6 Recovery Keywords

c-treeACE can completely recover from normal server shutdowns when data is under complete transaction processing control. These options control how recovery proceeds in various scenarios.

WARNING: Use caution when specifying options skipping files as they can result in data loss if you're not absolutely certain of the operation of your application regards to data file handling.

RECOVER_DETAILS (page 229)

Enables logging of detailed information about the c-treeACE automatic recovery process.

RECOVER_FILES (page 229)

Allows setting separate limits on the number of files used during automatic recovery and regular operations.

RECOVER_MEMLOG (page 230)

Loads one or more transaction logs into memory during automatic recovery to speed the recovery process.

RECOVER_SKIPCLEAN (page 230)

Causes files that appear to have been properly closed to not be updated during recovery.

RECOVER_TO_RESTORE_POINT (page 230)

Causes automatic recovery to recover to the last Restore Point.

REDIRECT (page 231)

Redirects filename references in the transaction logs during automatic recovery to the specified new filename.

REDIRECT_IFIL (page 231)

Specifies a file containing a list of files to be altered using the filename redirection rules specified with REDIRECT options.

SKIP_INACCESSIBLE_FILES (page 232)

Forces automatic recovery to skip any file that is not accessible.

SKIP_MISSING_FILES (page 232)

If a user file required during automatic recovery is missing, this keyword causes an error to be logged so the c-treeACE Server will successfully start up.

RECOVER_DETAILS

```
RECOVER_DETAILS <YES | NO>
```

This keyword enables logging of detailed information about the c-treeACE automatic recovery process. The time spent for each phase of automatic recovery in addition to the number of transactions processed for each phase is provided. This keyword adds minimal overhead to c-treeACE Server operations.

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

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

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

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

Default: YES

RECOVER_FILES

```
RECOVER_FILES <number of files | NO>
```

`RECOVER_FILES` makes it possible to set separate limits on the number of files used during automatic recovery and regular operations. The reason automatic recovery may require more files than regular operations is that during recovery files opened stay open until the end of

recovery. RECOVER_FILES takes as its argument the number of files to be used during recovery. If this is less than the number used during regular operation specified by the FILES keyword, the number of recovery files is set equal to the regular files and the keyword has no affect. If the number of recovery files is greater than the number of operational files, the number of files is adjusted downward at the end of automatic recovery freeing memory used by the additional control blocks, about 900 bytes per logical file.

Default: NO

RECOVER_MEMLOG

```
RECOVER_MEMLOG  <# of logs to load | NO>
```

Loads one or more transaction logs into memory during automatic recovery to speed the recovery process. The argument for this keyword specifies the maximum number of memory logs loaded into memory during automatic recovery.

Default: NO

RECOVER_TO_RESTORE_POINT

```
RECOVER_TO_RESTORE_POINT <YES | NO>
```

YES causes automatic recovery to recover to the last Restore Point.

When RECOVER_TO_RESTORE_POINT is YES, then automatic recovery (after a crash) comprises two steps:

1. the recovery of all transactions committed before the crash; and
2. the rollback of transactions to the last Active Restore Point.

If DELAYED_DURABILITY (page 216) is in effect at the time of the crash, then in step 1 it is not guaranteed that all transactions committed after the last Restore Point have their transaction log entries on disk (i.e., permanent storage).

Note: If DELAYED_DURABILITY (page 216) is in effect and RECOVER_TO_RESTORE_POINT is NO, then automatic recovery will attempt to recover all transactions that had committed before the crash; but some transactions committed after the Restore Point and before the crash may be recovered and others lost so that the files may be in an unexpected state. These is no way to predict which transactions may have been lost.

See Also

- KEEP_RESTORE_POINTS (page 218)

RECOVER_SKIPCLEAN

```
RECOVER_SKIPCLEAN <YES | NO>
```

This keyword may improve recovery times under certain circumstances. Files which appear to have been properly closed are not updated during recovery.

Note: For non-server systems, set the NINT global variable *ctskpclnfil* to 1 to enable this feature.

REDIRECT

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

One or more of the following configuration entries are used to specify redirection rules in the server configuration file *ctsrvr.cfg*:

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

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

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

Examples

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

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

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

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

Note: Use double quotes when a filename contains spaces.

See Also

REDIRECT_IFIL (page 231)

REDIRECT_IFIL

```
REDIRECT_IFIL <filename>
```

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

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

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

See Also

REDIRECT (page 231)

SKIP_INACCESSIBLE_FILES

```
SKIP_INACCESSIBLE_FILES YES
```

Forces automatic recovery to skip any file that is not accessible.

See Also

- SKIP_MISSING_FILES (page 232)
- SKIP_MISSING_LOG_MIRRORS (page 296)
- SKIP_MISSING_MIRRORS (page 296)

SKIP_MISSING_FILES

```
SKIP_MISSING_FILES <YES | NO>
```

This keyword is available for special c-treeACE Server startup conditions. If a user file required by the c-treeACE Server during automatic recovery was deleted, an error 12 might be returned and the c-treeACE Server would not continue. By adding `SKIP_MISSING_FILES` to the default *ctsrvr.cfg* file, the error will be logged and the c-treeACE Server will successfully start up.

`SKIP_MISSING_FILES` is *not* recommended as a permanent setting. Deleting files under transaction processing control adversely affects database integrity.

Default: NO

See Also

- SKIP_MISSING_LOG_MIRRORS (page 296)
- SKIP_MISSING_MIRRORS (page 296)
- SKIP_INACCESSIBLE_FILES (page 232)

11.7 File Management Keywords

AUTO_CLNIDXX (page 235)

Permits automatic **CLNIDXX()** during file open when a transaction high water mark, *hghtrn*, is found at file open to exceed the current transaction number.

AUTO_MKDIR (page 235)

When creating a c-tree data or index file, causes c-treeACE to automatically create directories for the filename that do not exist.

COALESCE_TRNLOG (page 236)

Attempt to combine deleted space with adjacent deleted space in ctTRNLOG files that do not have a *RECBYT* index.

FILE_CREATE_MODE (page 236)

Specifies the desired file permission mode assigned to files on Unix systems.

FILE_HANDLES (page 237)

On Unix, changes the number of file handles (at O/S level) that the O/S allows to be used by the c-treeACE Server.

FILE_PERMISSIONS (page 237)

Permits default file permissions to be assigned to one or more groups including two special groups: *WORLD* and *OWNER*.

INHERIT_FILE_PERMISSIONS (page 238)

Enables or disables inheriting file permissions from world to group to owner.

KEEPOPEN_CLOSE_RETRY_LIMIT (page 239)

Determines the number of times to retry the close operation before failing.

KEEPOPEN_LIST (page 239)

Upon file creation or physical open, attaches the *KEEPOPEN* attribute to the data file (and its indices) if the file is a data file that matches a <file spec> and the file creation/open is part of an ISAM creation or open.

MAX_VIRTUAL_FILES (page 241)

Specifies the maximum number of virtual files that may be opened at one time.

MEMFILE_MAX_BINS (page 241)

Sets the maximum number of hash bins allowed for a memory file at run time.

MEMORY_FILE (page 241)

Enables creating memory files if the file matches the specified file name.

SPLIT_NBR_OF_FILES (page 242)

Sets the number of files when using the partitioning file rule.

TMPNAME_PATH (page 242)

Sets the default directory for temporary files.

Compression

CMPREC_TYPE (page 235)

Specifies the type of data compression type for files.

COMPRESS_FILE (page 236)

Forces c-treeACE to create data files whose names match the specified name with data compression enabled.

Segmented Files

HUGE_TO_SEG_MB (page 238)

Force any huge file to be created as a segmented file.

MATCHING_SEGMENT (page 240)

Specifies behavior when an attempt to create a new file segment encounters an existing segment with the same name and the file ID matches the host file's ID and other attributes.

NONMATCHING_SEGMENT (page 241)

Specifies behavior when an attempt to create a new file segment encounters an existing segment with the same name and the file ID does not match the host file's ID and other attributes.

Diagnostics

Diagnostics keywords are intended to be used to help collect data to be analyzed for troubleshooting. The use of these keywords should be avoided in normal operation as they may cause additional overhead (e.g., logging).

DIAGNOSTICS PTADMIN (page 243)

Enables the filename and list of open instances of that file to be logged to *CTSTATUS.FCS* when a partition member purge fails with error 718.

AUTO_CLNIDXX

```
AUTO_CLNIDXX YES
```

Optionally permits automatic **CLNIDXX()** during file open when a transaction high water mark, *hghtrn*, is found at file open to exceed the current transaction number.

The disadvantage of this approach is that it is necessary to traverse all the index leaf nodes, writing those to disk that have been cleaned. For a very large index, this could be time consuming. Only customers that have experienced problems with **HTRN_ERR** (520) are candidates to use this configuration option. Also, internal, administrative requests to open the file read only (say as part of a dynamic dump or **ctFILBLK()** processing) are denied if the file is in the middle of an on the fly **CLNIDXX()**.

CTSTATUS.FCS receives messages concerning on the fly **CLNIDXX()**. Both success and failure are noted.

AUTO_MKDIR

```
AUTO_MKDIR  <YES | NO>
```

When creating a c-tree data or index file, c-treeACE can automatically create directories for the filename that do not exist.

Programmers can call the **Xtd8** file create function and set the *ctAUTOMKDIR* bit in the *splval* field of the *xcreblk* parameter to cause directories that do not exist to be created when creating that file.

Default: No

CMPREC_TYPE

```
CMPREC_TYPE  < TYPE >
```

Specifies the type of data compression type for files. Several algorithms are supported.

```
CMPREC_TYPE        < ZLIB | RLE | USER >
```

The following compression types are currently supported:

- *ZLIB* - General purpose compression from the zlib compression library.
- *RLE* - A fast proprietary run length encoding optimized for data that is nul character, space or zero (0) padded (sparse data files).

- *USER* - A user defined compression algorithm, requiring an associated .dll (or shared object)

With *USER* compression, these additional keywords must be entered in the configuration file in the order shown below, and a DLL name is required.

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

See Also

- COMPRESS_FILE (page 236, /doc/ctserver/#57549.htm)

COALESCE_TRNLOG

```
COALESCE_TRNLOG
```

Attempt to combine the deleted space with adjacent deleted space that already exists in *ctTRNLOG* files that do not have a *RECBYT* index (default is `OFF`). Combining deleted space can reduce fragmentation.

To enable this behavior, either:

- Add the server keyword `COALESCE_TRNLOG ON`

or

- Set the global variable `ctcoaltran = YES` in a standalone compile.

COMPRESS_FILE

```
COMPRESS_FILE  <filename>
```

Forces c-treeACE to create data files whose names match the specified name with data compression enabled. The file name may include wildcard characters (see **c-treeACE Standard Wildcards** (page 156)).

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

In V10.3 and later, the logic has been changed so that the `COMPRESS_FILE` keyword does not enable data compression for a data file that is created with resource support disabled. This prevents a rebuild from failing with **error 484** (could not open sort work file).

See Also

- **CMPREC_TYPE (page 235) in the** *c-tree Server Administrator's Guide***.**

FILE_CREATE_MODE

```
FILE_CREATE_MODE <mode>
```

On Unix systems, c-treeACE defaults to a permission mode of 0660 (read/write access for owner and group; no access for world) for the files it creates.

When using c-treeACE, the permission mode assigned to files can be set with the server configuration keyword `FILE_CREATE_MODE` to specify the desired file permission mode.

Example

```
;Set read and write permission for owner
;and no permission for group and world.
FILE_CREATE_MODE   0600
```

Note: On Unix systems, the system's umask setting also affects the permission mode assigned to a file when it is created. If the umask setting is non-zero, the specified permissions are removed from the file's permission mode.

FILE_HANDLES

```
FILE_HANDLES
```

This keyword is used on Unix system to change the number of file handles (at O/S level) that the O/S allows to be used by the c-treeACE Server.

No Default

FILE_PERMISSIONS

```
FILE_PERMISSIONS groupID#pmodeA#...#pmodeZ
```

Permits default file permissions to be assigned to one or more groups including two special groups: *WORLD* and *OWNER*. The primary need for this capability is to enforce permission flags on files that have already been created without a permission mask (i.e., the permission mask is zero at file create). A zero permission mask is equivalent to granting everyone all rights:

OPF_ALL | GPF_ALL | WPF_ALL

Note: *ALL* does not include the special *NOPASS* flag that permits a file to be opened for reading without supplying the file password. To grant *NOPASS* permission, it must be included explicitly.

- *groupID* is the name of a user group or the special groups WORLD and OWNER. The server does NOT verify that the groupIDs actually exist.
- *pmode* entries are symbolic names for the possible permission flags: *READ*, *WRITE*, *DEF*, *DELETE*, *ALL*, *NOPASS* and *NONE*. *NONE* should not be used with any other permission flags. It indicates no permissions are granted. Granting a permission of WRITE, *DEF* or *DELETE* is equivalent to granting all of the lesser permissions, thus *DELETE* and *ALL* are equivalent. For example, *#READ#WRITE* is equivalent to *#WRITE*. The *groupID* and *pmode* entries are case insensitive.

The *WORLD* entry applies to file opens by a user whose group(s) do not match any of the specified groups for those files without an explicit permission mask. If there is no WORLD entry, then WORLD permissions default to *ALL*. The *OWNER* entry applies to file opens by the users that created the files without an explicit permission masks.

Consider the following entries, and assume all the files in use did **not** have explicit permission masks at creation. Files with explicit permission masks (except for *OPF_ALL* | *GPF_ALL* | *WPF_ALL*) at creation are not affected by these `FILE_PERMISSIONS` entries.

```
FILE_PERMISSIONS   OWNER#DEF
FILE_PERMISSIONS   inventory#WRITE
FILE_PERMISSIONS   ACCOUNTING#write#nopass
FILE_PERMISSIONS   WORLD#NONE
```

In this example, the owner of a file will have *READ*, *WRITE* and *DEF* permissions. The owner of the file cannot delete the file. Members of the INVENTORY group have *READ* and *WRITE* permissions. Members of the ACCOUNTING group have *READ* and *WRITE* permissions and may open a file without its password (and receive *READ* permission only). A user who is not the owner of a file and not a member of the ACCOUNTING or INVENTORY groups will be assigned *WORLD* permissions, which in this case is *NONE*. *NONE* means the file cannot be opened.

If a user belongs to multiple groups, and two or more of its groups are specified with the `FILE_PERMISSIONS` keyword, the permissions granted to the user will be the union of the individual group permissions. However, if the user is the *OWNER* of the file, it receives *OWNER* permissions that default to *ALL*.

HUGE_TO_SEG_MB

```
HUGE_TO_SEG_MB  <segment size in MBs> [#<maximum number of segments>]
```

Force any huge file to be created as a segmented file.

On systems that do not support files greater than 2 GB or 4 GB, c-treeACE can still support huge files by creating tables as segmented files. The size of each segment stays below the OS maximum file size limit, but the aggregate size exceeds the limit.

The maximum number of segments is optional and defaults to sixteen (16). For example, to specify a segment size of 1 GB, and a maximum of 8 segments for a total file size of up to 8 GB, an entry would look like

```
HUGE_TO_SEG_MB  1024#8
```

If the file has been created with a maximum size in the *XCREblk* structure (see parameters for the extended create file function), then the number of segments will be computed to accommodate the maximum size.

Note: If dynamic dumps are used, then it would be appropriate to use the *!EXT_SIZE* script option so that the dump stream file would also be broken into automatic segments.

INHERIT_FILE_PERMISSIONS

```
INHERIT_FILE_PERMISSIONS YES | NO
```

By default, c-treeACE file permissions are inherited from world to group to owner. For example, if the **SECURITY()** function is called with a mode of *SEC_FILEMASK* and a permission mask of *WPF_READ*, *GPF_READ* and *OPF_READ* permissions are turned on. However, note that if *GPF_NONE* is specified, the permissions are not passed on to the group (and to the owner).

Note: Changing this option does not affect inheritance of permissions on files whose permissions have already been set. It only affects the inheritance of permissions when they are set at file create time or by calling the **SECURITY()** function with a mode of *SEC_FILEMASK*.

Default: **YES**

KEEPOPEN_CLOSE_RETRY_LIMIT

```
KEEPOPEN_CLOSE_RETRY_LIMIT  <limit>
```

Determines the number of times to retry the close operation before failing.

Default: 3

See Also

- KEEPOPEN_LIST (page 239)

KEEPOPEN_LIST

```
KEEPOPEN_LIST  <file spec>
```

<file spec> can be a file name or a partial name including wildcard characters. See **c-treeACE Standard Wildcards** (page 156). Upon file creation or physical open, (1) if the file name matches a <file spec> and (2) if the file is a data file and (3) if the data file creation/open is part of an ISAM creation or open, then the *KEEPOPEN* attribute is attached to the data file and its indices.

Memory files have the option of staying open after all users have closed the file. The motivation for keeping memory files open is to avoid losing the contents of the file so that subsequent users can open the file and read and/or update the contents.

When a non-memory file is physically closed, c-tree removes the data cache and/or index buffer entries associated with the file. A file is physically closed when all users that have the file open close the file. For a non-memory file, keeping the file open even after all users have closed the file permits the associated cache/buffer entries to stay in memory. Then subsequent opens have the benefit of the cache contents.

ISAM data files and their associated indices can also be designated as *KEEPOPEN* files. The motivation is to keep the files in the data cache and index buffers. It also eliminates a physical open when the next user opens the file. If all users have closed a *KEEPOPEN* file, then **ctCLSNAM()** can be called to close the data file and associated indices.

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

Note: If a file is kept open by `KEEPOPEN_LIST`, a call to **ctFILBLK** does not block access to it. Calling **CloseCtFileByName** will close the file and it will remain closed until unblocked by a subsequent **ctFILBLK** call. (The *c-treeACE Developer Guide http://docs.faircom.com/doc/ctreeplus* explains the **ctFILBLK** and **CloseCtFileByName** functions.)

Behavior Change for V10.3.1 and Later

For a data or index file that does not use full transaction logging (*ctTRNLOG*), c-treeACE Server now flushes updated cache pages and sets the update flag to zero for the file when the last user closes the file and the `KEEPOPEN_LIST` option keeps the file open.

The configuration option `KEEPOPEN_FLUSH NO` can be used in ***ctsrvr.cfg*** to disable this flushing behavior.

The is a change from previous behavior: When c-treeACE Server's `KEEPOPEN_LIST` configuration option was in effect for a c-tree data or index file that was not under transaction control, c-treeACE Server kept that file open after the last user closed the file. Updated data and index cache pages remained and were not written to disk before the call to close the file returned to the caller. This caused unnecessary data integrity risk should the c-treeACE Server abnormally terminate while the file remained open. The file is likely to fail to open with an **FCRP_ERR** error, 14, under such scenarios.

See Also

- KEEPOPEN_CLOSE_RETRY_LIMIT (page 239)

MATCHING_SEGMENT

```
MATCHING_SEGMENT <RENAME | DELETE | OVERWRITE | ERROR>
```

The segmented file logic behaved as follows when an attempt to create a new file segment encountered an existing segment with the same name:

- If the existing segment matched the host file's file ID and other attributes, then the existing segment was simply overwritten.
- If the file ID did not match, then an error was returned and no more records could be added to the file.

Overwriting a matching segment could cause a problem because the existing data is simply left in place until new records overwrite the data.

In addition to different defaults, the behavior for a matching segment (i.e., segment's file ID matches the host file's unique ID), is quite different than for a non-matching segment. For a matching segment, if `RENAME` is in effect and the rename fails, then a delete is attempted. If a delete fails, then the segment is overwritten. The error option must be explicitly requested. For a non matching segment, if either rename or delete are in effect and they fail, then an error occurs (and overwrite is not an option).

CTSTATUS.FCS entries are made when one of these unexpected segments are encountered. When a rename occurs, the segment's name is modified by adding the hexadecimal representation of the system's time in seconds to the end of the file name. For example,

```
mydata.s04
```

might become

```
mydata.s04.4465f728
```

where 0x4465f728 is the system time in seconds from some arbitrary starting point.

Default: RENAME

See Also

NONMATCHING_SEGMENT (page 241)

MAX_VIRTUAL_FILES

```
MAX_VIRTUAL_FILES  <Maximum files>
```

An integer argument specifying the maximum number of virtual files that may be opened at one time.

A *<Maximum files>* value of -1 forces all files as *ctPERMANENT*.

Default: 500

MEMFILE_MAX_BINS

```
MEMFILE_MAX_BINS <bins>
```

The maximum size of a memory file determines the number of hash bins that c-treeACE allocates for the memory file. However, there is a hard limit on the number of hash bins for a memory file. Previously, this limit was set at compile time to 65536. This configuration option is used to set the maximum number of hash bins allowed for a memory file at run time. This may be necessary for efficient access to very large memory files.

Default: 65537

MEMORY_FILE

The c-treeACE Server supports creating memory files using a server configuration keyword. This feature allows developers to create memory files using their existing application code, provided that the file is created using an *Xtd8* create function such as **CreateIFileXtd8()**. To create a memory file using the server configuration keyword, specify one or more entries of the form:

```
MEMORY_FILE <file name>#<max size>
```

where the file name may include wildcard characters (see **c-treeACE Standard Wildcards** (page 156)) and the maximum size is optional. If no maximum size is specified, then 4GB is used. If a file is being created and matches one of the `MEMORY_FILE` file name entries, then it will be created as a memory file unless it is a superfile host, superfile member, mirrored, segmented or partitioned file.

To cause all possible files to be created as memory files, add the following configuration entry:

```
MEMORY_FILE *
```

The `MEMORY_FILE` keyword is useful to quickly test how a file or set of files will behave as memory files.

NONMATCHING_SEGMENT

```
NONMATCHING_SEGMENT  <RENAME | DELETE | ERROR>
```

The segmented file logic behaved as follows when an attempt to create a new file segment encountered an existing segment with the same name:

- If the existing segment matched the host file's file ID and other attributes, then the existing segment was simply overwritten.
- If the file ID did not match, then an error was returned and no more records could be added to the file.

Overwriting a matching segment could cause a problem because the existing data is simply left in place until new records overwrite the data.

In addition to different defaults, the behavior for a matching segment (i.e., segment's file ID matches the host file's unique ID), is quite different than for a non matching segment. For a matching segment, if `RENAME` is in effect and the rename fails, then a delete is attempted. If a delete fails, then the segment is overwritten. The error option must be explicitly requested. For a non matching segment, if either rename or delete are in effect and they fail, then an error occurs (and overwrite is not an option).

CTSTATUS.FCS entries are made when one of these unexpected segments are encountered. When a rename occurs, the segment's name is modified by adding the hexadecimal representation of the system's time in seconds to the end of the file name. For example,

```
mydata.s04
```

might become

```
mydata.s04.4465f728
```

where 0x4465f728 is the system time in seconds from some arbitrary starting point.

Default: ERROR

See Also

`MATCHING_SEGMENT` (page 240)

SPLIT_NBR_OF_FILES

```
SPLIT_NBR_OF_FILES  <# of files>
```

When using the partitioning file rule:

```
<file_part_ind> = ( <int_unique_key> % <nbr_of_files> ) + 1
```

<nbr_of_files> is configured by the `SPLIT_NBR_OF_FILES` configuration entry.

> **Note:** Partitioning is currently only supported with a custom build containing the partition rule. Please contact your nearest FairCom office for availability.

TMPNAME_PATH

```
TMPNAME_PATH <path>
```

The `TMPNAME_PATH` location becomes the default directory for temporary files. On startup, if a `TMPNAME_PATH` entry is encountered in *ctsrvr.cfg*, the c-treeACE Server tests the validity of the path. If there is an error, the c-treeACE Server terminates. Whether or not successful, the c-treeACE Server enters the path name in the *CTSTATUS.FCS* file.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: Current server directory

DIAGNOSTICS PTADMIN

```
DIAGNOSTICS PTADMIN
```

When a partition member purge fails with error 718, this enables the filename and list of open instances of that file to be logged to *CTSTATUS.FCS*. Below is an example. For each connection we list the task ID, user name, node name, and user file number.

```
Mon Dec 12 12:40:33 2011
 - User# 00012  PT_ADMIN: purge failed, partition .\ctreesql.dbs\admin_pt.20111129.015307.dat is
open (2 open instances):
Mon Dec 12 12:40:39 2011
 - User# 00012  PT_ADMIN: Connection 16: ADMIN(SQL:CTREESQL) 64
Mon Dec 12 12:40:44 2011
 - User# 00012  PT_ADMIN: Connection 17: ADMIN(SQL:CTREESQL) 45
```

A process stack is also created when this occurs. This option can also be enabled via **SETCONFIG()** and through the **ctadmn** utility.

11.8 Lock Keywords

AUTO_LOCK_RETRY (page 244)

Enables automatic retries of data record locks when a c-tree function call fails with a data record lock error **DLOK_ERR**.

AUTO_LOCK_RETRY_SLEEP (page 245)

Set the sleep time between retries.

BLOCKING_LOCK_TIMEOUT_SEC (page 245)

Avoids excessively long blocking lock waits by returning error **UTIM_OUT** (827) to the caller of the lock request.

ITIM_RETRY_DEFER (page 245)

Introduces a delay before internally retrying an ISAM record read operation when an **ITIM_ERR** (160) is encountered.

ITIM_RETRY_LIMIT (page 246)

Enables retrying the operation the specified number of times before returning **ITIM_ERR** (160) if an index is out-of-sync with the data temporarily while another user is performing an update.

Diagnostics

Diagnostics keywords are intended to be used to help collect data to be analyzed for troubleshooting. The use of these keywords should be avoided in normal operation as they may cause additional overhead (e.g., logging).

DIAGNOSTICS DLOK_ERR (page 246)

Enables logging of **DLOK_ERR** (42) lock error information to *CTSTATUS.FCS*.

DIAGNOSTICS LOCK_DUMP (page 246)

Enable the use of the **LockDump()** function for non ADMIN users.

AUTO_LOCK_RETRY

```
AUTO_LOCK_RETRY <retries>
```

Enables automatic retries of data record locks when a c-tree function call fails with a data record lock error **DLOK_ERR**.

Default: 0 (retries).

AUTO_LOCK_RETRY_SLEEP

`AUTO_LOCK_RETRY_SLEEP <milliseconds>`

Set the sleep time between retries.

Default: 100 ms

BLOCKING_LOCK_TIMEOUT_SEC

`BLOCKING_LOCK_TIMEOUT_SEC <timeoutSEC>`

`BLOCKING_LOCK_TIMEOUT_SEC` avoids excessively long blocking lock waits. When used, this feature returns error **UTIM_OUT** (827) to the caller of the lock request. The function **ctLOKTIMOUT()** is used to set, change and clear this timeout feature.

- *timeoutSEC* is specified in seconds, and determines the amount of time to wait before returning the **UTIM_OUT** error.

This configuration entry is equivalent to a member of the ADMIN group making the call **ctLOKTIMOUT(***-1*, *ctLTOallusers* | *ctLTOdiagnostic*, *timeoutSEC***)**.

> **Note:** The effect of the configuration entry can be turned off by a call of the form **ctLOKTIMOUT(***-1*, *ctLTOallusers*, *0***)**.

If a user calls **ctLOKTIMOUT()** with a *datno* equal to -1 in order to set a timeout value on all data files for the user, the user can selectively change or turn off the timeout by making additional calls to **ctLOKTIMOUT()** specifying the data file number.

Lock Statistics

Locking statistics have an inconsistency when a lock request is removed from a list of waiting lock requests. When a lock request times out with this new feature, it is removed from the wait list. For instance, if a thread is waiting for a lock and it is killed by **ctadmn**, the lock is removed from the waiting list, however, the lock statistics do not reflect this. In fact, the count of currently blocked locks will be off (too high) by one for each lock request removed from a wait list. A new lock statistic has been added to account explicitly for lock requests that have been removed from the wait list: “killed.” It is treated in the same manner as the deadlock category.

ITIM_RETRY_DEFER

`ITIM_RETRY_DEFER <defer_time>`

Introduces a delay before internally retrying an ISAM record read operation when an **ITIM_ERR** (160) is encountered. `<defer_time>` specifies the time in milliseconds for which the server sleeps a thread that encounters error **ITIM_ERR** during an ISAM record read operation before retrying the ISAM record read operation. As before, the maximum number of **ITIM_ERR** retries for a particular ISAM record read operation is determined by the `ITIM_RETRY_LIMIT` configuration option.

`ITIM_RETRY_DEFER -1` disables the delay between **ITIM_ERR** retries.

Comments

If you frequently encounter ISAM record read operations failing with error **ITIM_ERR**, consider what the error code is revealing about your application's design: if you properly use record locking, **ITIM_ERR** can still occur if many users are updating the same records in sequence (that is, each of a number of threads reads the record with a lock, updates the record--changing the key value--and unlocks the record). In this case, increasing the `ITIM_RETRY_LIMIT` and `ITIM_RETRY_DEFER` settings can help avoid the **ITIM_ERR** errors, however, this may be at the expense of application performance, by introducing additional retries and delays between retries. In such a situation, consider ways to change the application to reduce the number of users that are attempting to update the same records at the same time.

Default: -1

See Also

`ITIM_RETRY_LIMIT (page 246)`

ITIM_RETRY_LIMIT

```
ITIM_RETRY_LIMIT <# of retries before returning ITIM_ERR>
```

In a multi-user environment, it is possible that an index may become out of sync with the data temporarily while another user is performing an update operation. Typically, error **ITIM_ERR** (160) is returned under these circumstances. It may be practical in some situations to retry the operation as the index is eventually updated. To accomplish this, the **ITIM_ERR** retry limit is configurable.

In addition, *SNAPSHOT* output now contains the value of the limit and the number of failed retry loops. Before this change, only the number of retries was reported by the *SNAPSHOT*.

Default: Typically 10 or 20

See Also

`ITIM_RETRY_DEFER (page 245)`

DIAGNOSTICS DLOK_ERR

```
DIAGNOSTICS DLOK_ERR
```

Enables logging of **DLOK_ERR** (42) lock error information to *CTSTATUS.FCS*.

The log entry shows the filename, data record offset, and the lock owner.

Example output:

```
Mon Jun 05 15:18:42 2006
 - User# 00012 DLOK_ERR: file=vcusti offset=0x0000-00002899 owner=11
```

DIAGNOSTICS LOCK_DUMP

```
DIAGNOSTICS LOCK_DUMP
```

Enable the use of the **LockDump()** function for non ADMIN users. The ADMIN user account can call the **LockDump()** function without this keyword enabled.

Default: Disabled

11.9 I/O Keywords

COMPATIBILITY DIRECT_IO (page 249)

Reverts to the previous behavior of using direct I/O instead of O_DSYNC synchronous writes (or O_SYNC if O_DSYNC is not defined) for transaction logs on Solaris.

COMPATIBILITY FDATASYNC (page 250)

For Unix systems, enables the optional use of **fdatasync()** instead of **fsync()**.

COMPATIBILITY FORCE_WRITETHRU (page 250)

Forces the automatic addition of the *WRITETHRU* mode to all files opened without the *TRNLOG* file mode.

COMPATIBILITY PREV610A_FLUSH (page 250)

Provides a balance between update performance and recoverability of data in the event of an abnormal c-treeACE Server termination affecting non-transaction files.

COMPATIBILITY WTHRU_UPDFLG (page 251)

Disables the ‘update’ flag on files with the *WRITETHRU* file mode.

DEFAULT_CHANNELS (page 251)

Changes the number of I/O channels assigned to a file with *ctDUPCHANEL* in its file mode at open, unless the file is in the `SET_FILE_CHANNELS` list.

IDLE_NONTRANFLUSH and IDLE_TRANFLUSH (page 251)

Sets the interval c-treeACE waits before checking to see if the server is idle before flushing data and index caches during idle time.

IO_ERROR_BLOCK_RETRY (page 252)

Specifies the maximum number of failed `IO_ERROR_BLOCK_SIZE`-sized I/O operations that must occur before the I/O operation is considered to have failed.

IO_ERROR_BLOCK_SIZE (page 253)

Causes a read or write operation that fails with Windows system error 1450 (ERROR_NO_SYSTEM_RESOURCES) to be retried in blocks of the specified size.

IO_ERROR_BLOCK_SLEEP (page 253)

Specifies a time in milliseconds between `IO_ERROR_BLOCK_RETRY` retry attempts.

SET_FILE_CHANNELS (page 253)

Permits the number of I/O channels to be explicitly set for the named file regardless of whether the file mode, at open, includes *ctDUPCHANEL*.

UNBUFFERED_IO (page 254)

Enables unbuffered I/O for the specified file on Windows systems.

Diagnostics

Diagnostics keywords are intended to be used to help collect data to be analyzed for troubleshooting. The use of these keywords should be avoided in normal operation as they may cause additional overhead (e.g., logging).

DIAGNOSTICS DIRECT_IO (page 255)

Enables a check on each write operation for a file for which direct I/O is requested that the properties of the write operation meet the direct I/O requirements.

DIAGNOSTICS LOWL_FILE_IO (page 255)

Logs low-level system errors into the server status file, *CTSTATUS.FCS*.

COMPATIBILITY DIRECT_IO

```
COMPATIBILITY DIRECT_IO
```

The c-treeACE Server for Solaris uses O_DSYNC synchronous writes for the transaction logs when the `COMPATIBILITY SYNC_LOG` or `COMPATIBILITY LOG_WRITETHRU` configuration options are specified in the server configuration file.

Furthermore, it is expected the O_DSYNC writes are more efficient than O_SYNC writes. The c-treeACE Server uses O_DSYNC on systems that define this file open mode and uses O_SYNC on systems that do not define the O_DSYNC file open mode.

`COMPATIBILITY DIRECT_IO`, is available to revert to the previous behavior of direct I/O.

When the c-treeACE Server starts up, it writes either the message "Transaction logs using direct I/O." or "Transaction logs using synchronous I/O." to *CTSTATUS.FCS* to indicate which method of synchronous writes is in use. If neither `COMPATIBILITY SYNC_LOG` nor `COMPATIBILITY LOG_WRITETHRU` is specified in *ctsrvr.cfg*, neither of these messages is written to *CTSTATUS.FCS* as in that case the server performs asynchronous writes that are flushed by a separate call.

Default: OFF

Also See

```
COMPATIBILITY LOG_WRITETHRU (page 214)
COMPATIBILITY SYNC_LOG (page 327)
```

COMPATIBILITY FDATASYNC

```
COMPATIBILITY FDATASYNC
```

For Unix systems, enables the optional use of **fdatasync()** instead of **fsync()**.

COMPATIBILITY FORCE_WRITETHRU

```
COMPATIBILITY FORCE_WRITETHRU
```

Forces the automatic addition of the *WRITETHRU* mode to all files opened without the *TRNLOG* file mode.

Default: Pre-V11: Not present.

Beginning with V11:

The following keywords are now specified in default c-treeACE Server configuration files:

```
COMPATIBILITY FORCE_WRITETHRU
COMPATIBILITY PREV610A_FLUSH
```

`COMPATIBILITY FORCE_WRITETHRU` enables the *WRITETHRU* filemode for all non-transaction files (which includes *PREIMG* files). The performance impact of *WRITETHRU* depends on whether `COMPATIBILITY PREV610A_FLUSH` is also enabled:

- When `COMPATIBILITY PREV610A_FLUSH` is NOT enabled, each update to a non-transaction file is flushed to disk. Both c-tree and operating system (OS) caches are flushed. This mode, while extremely safe, does negatively impact performance.
- When `COMPATIBILITY PREV610A_FLUSH` is enabled, each update to a non-transaction file is flushed to the OS file system cache, however *the OS file system is not flushed*. Updated data is potentially vulnerable in this state. However, this mode performs extremely fast.

Be sure to understand the impact of these file modes with respect to your file transaction mode in use, and type and vulnerability of your application data.

COMPATIBILITY PREV610A_FLUSH

```
COMPATIBILITY PREV610A_FLUSH
```

This options provides a good balance between update performance and recoverability of data in the event of an abnormal c-treeACE Server termination and affects non-transaction files as follows:

- non-transaction files that are neither created nor opened with the *WRITETHRU* filemode have their updates written to the c-treeACE Server cache and eventually written to the file system cache and to disk.
- non-transaction files that are created or opened with the *WRITETHRU* filemode have their updates flushed to the file system cache.

Beginning with V11:

The following keywords are now specified in default c-treeACE Server configuration files:

```
COMPATIBILITY FORCE_WRITETHRU
```

```
COMPATIBILITY PREV610A_FLUSH
```

`COMPATIBILITY FORCE_WRITETHRU` enables the *WRITETHRU* filemode for all non-transaction files (which includes *PREIMG* files). The performance impact of *WRITETHRU* depends on whether `COMPATIBILITY PREV610A_FLUSH` is also enabled:

- When `COMPATIBILITY PREV610A_FLUSH` is NOT enabled, each update to a non-transaction file is flushed to disk. Both c-tree and operating system (OS) caches are flushed. This mode, while extremely safe, does negatively impact performance.
- When `COMPATIBILITY PREV610A_FLUSH` is enabled, each update to a non-transaction file is flushed to the OS file system cache, however *the OS file system is not flushed*. Updated data is potentially vulnerable in this state. However, this mode performs extremely fast.

Be sure to understand the impact of these file modes with respect to your file transaction mode in use, and type and vulnerability of your application data.

COMPATIBILITY WTHRU_UPDFLG

```
COMPATIBILITY WTHRU_UPDFLG
```

Disables the 'update' flag on files with the *WRITETHRU* file mode. If `COMPATIBILITY WTHRU_UPDFLG` is not in its configuration file and if non-transaction files are opened without *WRITETHRU* in the file mode, a warning is issued in *CTSTATUS.FCS* concerning the vulnerability to **FCRP_ERR** (14).

Default: Not present

DEFAULT_CHANNELS

```
DEFAULT_CHANNELS   <nbr of I/O channels>
```

`DEFAULT_CHANNELS` changes the number of I/O channels assigned to a file with *ctDUPCHANEL* in its file mode at open, unless the file is in the `SET_FILE_CHANNELS` list. The default number of channels is not limited by the NUMCHANEL value.

> **Note:** Multiple I/O channels are disabled for newly created files. The multiple I/O channels take affect only on an open file call. Also, depending on the default number of I/O channels, a superfile host not in the `SET_FILE_CHANNELS` will use no more than 2 * NUMCHANEL I/O channels.

See Also

`SET_FILE_CHANNELS (page 253)`

IDLE_NONTRANFLUSH and IDLE_TRANFLUSH

```
IDLE_TRANFLUSH <idle check interval seconds>
IDLE_NONTRANFLUSH <idle check interval seconds>
```

c-treeACE flushes data and index caches during idle time, launching two idle thread processes at start-up: One thread flushes transaction-file buffers and the other flushes non-transaction-file buffers. The threads wake-up periodically and check if the c-treeACE Server is idle to begin flushing. Subsequent activity terminates the flushes. Low priority background threads, such as the

delete node thread, do not affect the idle state, however, c-treeACE clients and transaction checkpoints modify the idle status.

Filesystem Flush Performance

Starting with V8 the default behavior of the idle flush threads was changed to skip internal fiesystem flush calls for better performance when performing the idle flush operation. Flushing the filesystem buffers guarantees data is secured to disk at a performance cost. To revert this behavior, add the #SAVE option after the checkpoint time which will ensure data safety in nearly all cases. Note that flushing filesystem buffers will cause measurable delays when large caches are in use, and you may notice transaction slowdowns during this period.

```
IDLE_TRANFLUSH <idle check interval in seconds>#SAVE
IDLE_NONTRANFLUSH <idle check interval in seconds>#SAVE
```

The default interval is 15 seconds. Setting the interval to zero or a negative value disables the thread.

Default: 15 seconds

See Also

- DELAYED_DURABILITY (page 216)

IO_BLOCK_SIZE

```
IO_BLOCK_SIZE <size>
```

Splits disk read and write operations larger than the specified size into individual operations of the specified size. For example, specifying `IO_BLOCK_SIZE 16 KB` causes a 1 MB write request to be performed as 64 16 KB write operations.

See Also

- IO_ERROR_BLOCK_RETRY (page 252)
- IO_ERROR_BLOCK_SIZE (page 253)
- IO_ERROR_BLOCK_SLEEP (page 253)

IO_ERROR_BLOCK_RETRY

```
IO_ERROR_BLOCK_RETRY <retries>
```

Specifies the maximum number of failed `IO_ERROR_BLOCK_SIZE`-sized I/O operations that must occur before the I/O operation is considered to have failed. If the `IO_ERROR_BLOCK_SIZE`-sized I/O operations that are being attempted for a particular I/O operation fail more than *<retries>* times, the c-treeACE Server writes a **READ_ERR** (36) or **WRITE_ERR** (37) message to *CTSTATUS.FCS* and considers the I/O operation to have failed.

A value of -1 signifies infinite retries. The default is 0, which means that the I/O operation is tried only once in IO_ERROR_BLOCK_SIZE-sized blocks, and if any of these I/O operations fails, the entire I/O operation is considered to have failed. As another example, if IO_ERROR_BLOCK_RETRY is set to 20 and IO_ERROR_BLOCK_SIZE is set to 65536, if a

327680-byte write is retried as 5 65536-byte write operations, then the I/O operation fails if there are 20 failures to perform those 5 write operations.

See Also

- IO_ERROR_BLOCK_SIZE (page 253)
- IO_ERROR_BLOCK_SLEEP (page 253)

IO_ERROR_BLOCK_SIZE

```
IO_ERROR_BLOCK_SIZE <size>
```

When the Windows kernel has allocated all of its paged-pool memory, it will not be able to perform many tasks and instead returns a STATUS_INSUFFICIENT_RESOURCES (0xC000009A) message. This is a restriction of 32-bit addressing (only 2GB addressable within the kernel), regardless of the amount of memory available in the system.

Microsoft Support Knowledgebase regarding Error 1450 http://support.microsoft.com/kb/142719

When the c-treeACE Server configuration option `IO_ERROR_BLOCK_SIZE` option is specified in the c-treeACE Server configuration file, a read or write operation that fails with Windows system error 1450 (ERROR_NO_SYSTEM_RESOURCES) is retried in blocks of the specified size. If any one of those read or write operations fails, the c-treeACE Server fails the read or write operation.

See Also

- IO_ERROR_BLOCK_RETRY (page 252)
- IO_ERROR_BLOCK_SLEEP (page 253)

IO_ERROR_BLOCK_SLEEP

```
IO_ERROR_BLOCK_SLEEP <time>
```

Specifies a time in milliseconds between `IO_ERROR_BLOCK_RETRY` retry attempts. The default is zero, which means that retries are attempted immediately.

See Also

- IO_ERROR_BLOCK_SIZE (page 253)
- IO_ERROR_BLOCK_RETRY (page 252)

SET_FILE_CHANNELS

```
SET_FILE_CHANNELS  <file name>#<nbr of I/O channels>
```

Without this feature, the *ctDUPCHANEL* file mode bit enables a file to use NUMCHANEL simultaneous I/O channels, where NUMCHANEL is set at compile time, and is by default, set to two (2). For superfile hosts with *ctDUPCHANEL*, 2* NUMCHANEL I/O channels are established. The default is not to turn on ctFeatCHANNELS.

SET_FILE_CHANNELS permits the number of I/O channels to be explicitly set for the named file regardless of whether the file mode, at open, includes *ctDUPCHANEL*. A value of one (1) for the number of I/O channels effectively disables *ctDUPCHANEL* for the file. A value greater than one (1) turns on DUPCHANEL and determines the number of I/O channels used. The number of I/O channels is not limited by the compile-time NUMCHANEL value. You may have as many SET_FILE_CHANNELS entries as needed.

The *<file name>* can be a wildcard specification using a '?' for a single character and a '*' for zero or more characters. See **c-treeACE Standard Wildcards** (page 156).

Note: Multiple I/O channels are disabled for newly created files. The multiple I/O channels take affect only on an open file call. Also, depending on the default number of I/O channels, a superfile host not in the SET_FILE_CHANNELS will use no more than 2 * NUMCHANEL I/O channels.

See Also

DEFAULT_CHANNELS (page 251)

UNBUFFERED_IO

```
UNBUFFERED_IO  <filename>
```

c-treeACE Server supports the use of unbuffered disk I/O operations on a per-file basis. Unbuffered I/O bypasses the file system cache and avoids double caching of data in the c-treeACE Server and the file system cache. The file name may include wildcard characters (see **c-treeACE Standard Wildcards** (page 156))

The UNBUFFERED_IO configuration option enables unbuffered I/O for the specified file. When the file is opened, the sector size of the disk on which the file resides is determined and it stores that sector size in the new file control block member.

Windows enforces the following restrictions for I/O when using unbuffered I/O:

1. The file offset for the I/O operation must be a multiple of the disk sector size.
2. The amount of data to be read or written must be a multiple of the disk sector size.
3. The address of the buffer used in the I/O operation must be aligned on a disk sector size boundary.

For files that use unbuffered I/O, c-tree's file I/O function checks that these requirements are met. If not, logic is in place that makes the necessary adjustment by allocating a temporary buffer that is used in the I/O operation.

When a file is configured to use unbuffered I/O, the sector size of the disk on which the file is stored is checked and if it exceeds the server's page size. If so, the following message is logged to *CTSTATUS.FCS* and unbuffered I/O is not used on that file:

If an existing file is being opened:

```
mbopen: File <filename> disk sector size (<disk_sector_size>) exceeds page size (<page_size>)
```

If a new file is being created:

```
mbcratx: File <filename> disk sector size (<disk_sector_size>) exceeds page size (<page_size>)
```

Note: Unbuffered I/O is not available for for encrypted or segmented files and is ignored for those file types.

`UNBUFFERED_IO` completely avoids the file system cache. Compare to COMPATIBILITY TDATA_WRITETHRU (page 215), which uses the file system cache and then to disk before returning.

OS Support

This option is supported on the Windows operating system.

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

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

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

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

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

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

See Also

- UNBUFFERED_LOG_IO (page 227)

DIAGNOSTICS DIRECT_IO

```
DIAGNOSTICS DIRECT_IO
```

Enables a check on each write operation for a file for which direct I/O is requested that the properties of the write operation meet the direct I/O requirements. If not, the server logs one of the following messages to the server status log:

```
directIOdiag: Buffer address (<address>) not properly aligned for direct I/O...
directIOdiag: Write length (<length>) not properly sized for direct I/O...
directIOdiag: File offset (<offset>) not properly aligned for direct I/O...
```

DIAGNOSTICS LOWL_FILE_IO

```
DIAGNOSTICS  LOWL_FILE_IO
```

The `DIAGNOSTICS LOWL_FILE_IO` server keyword logs low-level system errors into the server status file, *CTSTATUS.FCS*. Although client applications have access to system errors through *sysiocod*, it is useful to see these errors logged on the server side. Activating this feature causes a high decrease in performance.

This feature is also available at run time via the **ctSETCFG()** API call and passing in (*setcfgDIAGNOSTICS,* "LOWL_FILE_IO")

From the c-treeACE Server administrator utility, **ctadmn**, choose option 10, "Change Server Settings" and then option 8, "Change a DIAGNOSTICS option". Then enter LOWL_FILE_IO to enable. Precede with a '~' character to disable.

Default: Disabled

11.10 Logging and Monitoring Keywords

CHECKPOINT_MONITOR (page 259)

Determines if each occurrence of an internal c-treeACE Server checkpoint will cause a time stamp message to be sent to the c-treeACE Server console screen and to the *CTSTATUS.FCS* file.

CTSTATUS_MASK (page 259)

Allows certain types of entries in *CTSTATUS.FCS* to be suppressed.

CTSTATUS_SIZE (page 260)

Controls the size of the c-treeACE Server status file.

DISK_FULL_ACTION (page 261)

Enables c-treeACE to monitor available disk space and to shut down the database engine when the disk space falls below the specified limit.

DISK_FULL_LIMIT (page 262)

Activates a disk space threshold mechanism to detect when a disk volume is getting full (the specified number of bytes must remain available on a disk volume after a file has been extended).

DISK_FULL_VOLUME (page 262)

Allows volume-specific disk full checks.

FUNCTION_MONITOR (page 263)

Causes the client number, function number, function name, and file name are displayed in a scrolling fashion on the c-treeACE Server console screen.

LOCK_MONITOR (page 263)

Monitors the number of active record locks.

MEMORY_MONITOR (page 264)

Sends a message to the console whenever allocated memory exceeds the next memory threshold.

MEMORY_TRACK (page 264)

Sends debug output to the console every time the net memory allocation count changes by a multiple of the threshold value.

MONITOR_MASK (page 264)

Suppress the message sent to the system console if a file without a matching name does match the unique file ID when a file open is attempted.

PERF_MONITOR (page 264)

Allows entries to be placed on a queue for client side processing and monitoring of server events.

REQUEST_TIME_MONITOR (page 265)

Specifies a request time in seconds for monitoring function that exceed the specified time.

SNAPSHOT_FILENAME (page 265)

Captures user information pertaining the to file specified to *SNAPSHOT.FCS* in addition to the system snapshot.

SNAPSHOT_INTERVAL (page 266)

Enables automatic snapshots at specified intervals.

SNAPSHOT_LOCKWAIT_USEC (page 266)

Changes the default histogram intervals (box width) for the lock waiting time histograms.

SNAPSHOT_TRANTIME_USEC (page 266)

Changes the default histogram intervals (box width) for the transaction time histogram.

SNAPSHOT_USERID (page 266)

Captures user information to SNAPSHOT.FCS in addition to the system snapshot.

SYSLOG (page 267)

Specifies a keyword indicating contents to be stored in the System Event Log.

SYSVIEW_WHAT (page 267)

Enables system status reports triggered by the specified event.

SYSVIEW_WHEN (page 268)

Enables system status reports triggered by the specified event.

Diagnostics

Diagnostics keywords are intended to be used to help collect data to be analyzed for troubleshooting. The use of these keywords should be avoided in normal operation as they may cause additional overhead (e.g., logging).

DEADLOCK_MONITOR (page 261)

Causes a time stamp message goes to the c-treeACE Server console screen each time the c-treeACE Server detects and resolves a dead lock.

DIAGNOSTICS SNAPSHOT_AUTOMATIC (page 269)

Write system snapshots to the human-readable *SNAPSHOT.FCS* text file.

DIAGNOSTICS SNAPSHOT_IOTIME (page 269)

Enables collecting disk read and write timings on a per-file basis when high-resolution timer support is activated.

DIAGNOSTICS SNAPSHOT_SHUTDOWN (page 269)

Write system snapshots to the human-readable *SNAPSHOT.FCS* text file.

DIAGNOSTICS SNAPSHOT_WORKTIME (page 269)

Enables collecting c-treeACE function call counts and timings on a per-c-tree file basis.

CHECKPOINT_MONITOR

```
CHECKPOINT_MONITOR <YES | NO | DETAIL>
```

This keyword takes one of three arguments: YES, NO, and DETAIL. If YES, each occurrence of an internal c-treeACE Server checkpoint will cause a time stamp message to be sent to the c-treeACE Server console screen and to the *CTSTATUS.FCS* file. The checkpoint is a snapshot of the c-treeACE Server at an instance in time and is used during automatic recovery. The checkpoint provides for a measure of the system activity. The DETAIL argument causes six intermediate milestones to be output for each checkpoint in addition to the beginning and ending checkpoint messages. These intermediate outputs aid in analyzing how the checkpoint procedure interacts with applications. If there is no system activity, no checkpoints will occur. This keyword should be used for debugging purposes only since performance may be compromised.

Default: NO

CTSTATUS_MASK

```
CTSTATUS_MASK  <mask>
```

Allows certain types of entries in *CTSTATUS.FCS* to be suppressed.

Accepted *<mask>* values

DYNAMIC_DUMP_FILES	Suppress the logging of the names of the files backed up by a dynamic dump operation.
MATCH_FILE_ID	Suppresses "Matching file Ids" messages. Such messages can be output in two situations: • If the files share the same FairCom 12-byte file ID, but are different files in which case an open error should result. • If the files are the same file, but have been opened with different names (or paths).
WARNING_AUTO_TRNLOG	Suppresses the logging of the message: "WARNING: could not turn on AUTO_PREIMG/AUTO_TRNLOG for ... (filename)" when the AUTO_PREIMG or AUTO_TRNLOG configuration option matches the name of a c-tree index file that was not created with support for transaction control.
WARNING_FCRP_ERR	Eliminates **FCRPT_ERR** (14) errors in *CTSTATUS.FCS*.
WRITE_ERR	Suppresses WRITE_ERR messages. See DIAGNOSTICS WRITE_ERR_DUMP
VDP_ERROR	Suppresses logging of communications errors.

Default: Log all messages.

CTSTATUS_SIZE

```
CTSTATUS_SIZE <file_size | negative_file_size | 0>
```

CTSTATUS_SIZE controls the size of the c-treeACE Server status file. The argument to CTSTATUS_SIZE is the approximate maximum size in bytes for *CTSTATUS.FCS*. When this limit is reached, *CTSTATUS.FCS* is renamed to *T0000001.FCS* and a new status file is created. The *T#.FCS* file numbers increase each time the limit is reached, similar to the transaction log files, i.e., the next time the maximum size is reached, *CTSTATUS.FCS* is renamed to *T0000002.FCS*.

To limit the number of archived status logs, set a negative value for CTSTATUS_SIZE. Only *T0000001.FCS* will be kept, being replaced each time *CTSTATUS.FCS* is archived.

A value of 0, the default, allows a the file to expand to a size limited by the operating system and storage availability.

Default: -32000000 (None, permitting unlimited size, prior to V11)

DBENGINE_CHECK

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

```
DBENGINE_CHECK = 1
```

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

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

DEADLOCK_MONITOR

```
DEADLOCK_MONITOR <YES | NO>
```

If YES, each time the c-treeACE Server detects and resolves a dead lock situation, a time stamp message goes to the c-treeACE Server console screen.

> **Note**: This keyword is used primarily for debugging since this feature consumes additional overhead.

Default: NO

DISK_FULL_ACTION

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

Enables c-treeACE to monitor available disk space and to shut down the database engine when the disk space falls below the specified limit.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Windows Examples

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

Unix Examples

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

The keyword can be specified multiple times. The keyword is also independent of the DISK_FULL_LIMIT and DISK_FULL_VOLUME keywords.

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

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

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

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

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

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

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

See Also

- DISK_FULL_LIMIT (page 262)
- DISK_FULL_VOLUME (page 262)

DISK_FULL_LIMIT

```
DISK_FULL_LIMIT <bytes available>
```

c-treeACE Servers for Unix and Windows support the `DISK_FULL_LIMIT` keyword, which activates a disk space threshold mechanism to detect when a disk volume is getting full. The `DISK_FULL_LIMIT` configuration keyword takes as its argument the number of bytes that must remain available on a disk volume after a file has been extended. If the update operation fails, a message is written in *CTSTATUS.FCS* naming the file involved.

c-treeACE Servers that do not support this feature ignore the `DISK_FULL_LIMIT` keyword.

Note: When the disk if full the c-treeACE files cannot be extended. The c-treeACE operations that attempt to add or update data in a file and require the files to be extended will fail with specific errors. This diagnostic keyword is used to put supplemental information in the status file and alert the user.

Default: No disk full check

See Also

- DISK_FULL_VOLUME (page 262)
- DISK_FULL_ACTION (page 261)

DISK_FULL_VOLUME

```
DISK_FULL_VOLUME /path
```

Allows volume-specific disk full checks. `DISK_FULL_VOLUME` takes as its argument a concatenated volume name and limit. A path separator must occur between the volume name and the threshold limit, which may be zero.

In Unix this is in the form */name/<limit>*. The following example places a disk full threshold of one million bytes on the volume */home*:

```
DISK_FULL_VOLUME /home/1000000
```

From Windows the form of the argument is *<DRIVE>:\<limit>*. The following example places a 1MB threshold on drive E:

```
DISK_FULL_VOLUME e:\1048576
```

Note: When the disk if full the c-treeACE files cannot be extended. The c-tree operations that attempt to add or update data in a file and require the files to be extended will fail with specific errors. This diagnostic keyword is used to put supplemental information in the status file and alert the user.

Default: Off

See Also

- DISK_FULL_LIMIT (page 262)
- DISK_FULL_ACTION (page 261)

FUNCTION_MONITOR

`FUNCTION_MONITOR <YES | NO | file_name>`

If YES, the client number, function number, function name, and file name are displayed in a scrolling fashion on the c-treeACE Server console screen. Alternatively, the same information, along with the return value and error codes for each function, can be routed to a file by specifying a file name. This keyword should be used primarily for debugging since this feature consumes additional overhead.

Note: Activate the function monitor dynamically under the c-treeACE Server for Windows by selecting **View > Function Monitor Window**.

Default: NO

Notes:

`FUNCTION_MONITOR` can be enabled without restarting the server using the **ctadmn** command-line tool as follows:

1. Start the **ctadmn** tool and connect to the server.
2. Select the following from the menu:
   ```
   10 "Change Server Settings"
   ```
3. Select the following from the next menu:
   ```
   1  "Configure function monitor"
   ```
4. Enter *fmon.log* at the prompt:
   ```
   Enter the new function monitor value.
   Allowed values are: YES or NO or filename
   ```

Function monitor information will be written to the ctreesql process working directory in a text file named *fmon.log*.

When you are done, be sure to shut off function monitoring: Repeat the steps above, but disable function monitoring by entering `NO` in the final step.

LOCK_MONITOR

`LOCK_MONITOR <threshold value>`

Monitors the number of active record locks. This keyword, or the **SetOperationState()** function call, enables a lock monitor that indicates when the number of lock calls over unlocks equals a multiple of the threshold value, or if it goes below a threshold.

For example, `LOCK_MONITOR 100` sends a message to the console each time the number of lock calls over unlocks equals a multiple of 100. Likewise, if the number of unbalanced lock calls falls below these thresholds, a message goes to the console. Ordinarily, when the number of locks are in balance (i.e., excess locks over unlocks equals zero) no message is routed to the console unless a message indicating an excess of locks has already been sent to the console. If you wish the message to be sent whenever the number of excess locks equals zero, enter the threshold value as a negative. For example, `LOCK_MONITOR  -100`

Default: No lock monitor

MEMORY_MONITOR

```
MEMORY_MONITOR <Bytes | NO>
```

Sends a message to the console whenever allocated memory exceeds the next memory threshold. The parameter specifies a size in bytes. For example, `MEMORY_MONITOR 500000` sends a message every time memory consumption exceeds the next 500,000 byte range of memory. The message is also sent when memory usage decreases for each absolute memory block. This keyword should be used primarily for debugging, as there is some additional overhead for this feature.

Default: NO

MEMORY_TRACK

```
MEMORY_TRACK <allocation threshold value>
```

Sends debug output to the console every time the net memory allocation count changes by a multiple of the threshold value. The count is the number of memory allocation requests. See also `DIAGNOSTICS TRACK_LOGON`.

Default: 0 (indicates do not track)

MONITOR_MASK

```
MONITOR_MASK
```

When a file open is attempted, the c-treeACE 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. By default, if a file without a matching name does match the unique file ID c-treeACE sends a message to the system console indicating the names of the two files involved. Suppress this message by adding the following entry:

```
MONITOR_MASK  MATCH_FILE_ID
```

Default: Enabled

PERF_MONITOR

```
PERF_MONITOR <option>
```

The *SYSMON* feature allows entries to be placed on a queue for client side processing and monitoring of server events. This option allows event entries to be specified in the *ctsrvr.cfg* configuration file. The following options are supported:

```
ALL_EVENTS
CHECKPOINT
LOG_EXTENSION
LONG_TRANSACTION
```

When checkpoints are monitored, an entry is written to the SYSMON_PERF client-side queue for the beginning and end of each checkpoint. Log extensions similarly generate begin and end entries. A long transaction generates one entry.

ALL_EVENTS is equivalent to listing each individual option.

When one or more event types have been enabled, then the first entry written into the SYSMON_PERF queue provides the information necessary to convert the high-resolution times into seconds and/or dates.

See Also

- LONG_TRANSACTION_MS (page 221)

REQUEST_TIME_MONITOR

```
REQUEST_TIME_MONITOR <request_time_interval>
```

c-treeACE supports monitoring function request times that exceed a specified time limit with the REQUEST_TIME_MONITOR <request_time_interval> keyword. *<request_time_interval>* is a time in seconds. A value of -1 disables the feature, meaning that it cannot be dynamically turned on using the **ctSETCFG()** function, as described below. A non-negative value (the minimum value is 10) specifies a request time in seconds. An internal thread checks for requests whose time has exceeded the specified request time. When the thread finds such a request, it writes a message to *CTSTATUS.FCS*, creates a process stack dump (on systems that support this feature), and logs diagnostic information to the file *RQSTMON.FCS.*

The **ctSETCFG()** function can be used to change the request time monitor interval value. For example:

```
ctSETCFG( setcfgREQUEST_TIME_MONITOR, "60" );
```

sets the request time interval to 60 seconds. If the caller of **ctSETCFG()** is not a member of the ADMIN group, **ctSETCFG()** returns error **LADM_ERR** (589, member of ADMIN group required). Attempting to specify a negative value returns **PBAD_ERR** (749, bad parameter value). Attempting to enable this feature when REQUEST_TIME_MONITOR -1 is specified in the configuration file returns **SETO_ERR** (804, cannot override configuration option).

The previously unused field *scttrnavl* in the system snapshot structure (*ctGSMS*) has been renamed *sctrqtmonint* and is set to the request time monitor interval value. This change resulted in the system snapshot structure version change from 7 to 8.

Default: NO

SNAPSHOT_FILENAME

```
SNAPSHOT_FILENAME   <file name>
```

By default, only the system snapshot is captured with SNAPSHOT_INTERVAL. SNAPSHOT_FILENAME is used to capture user information to *SNAPSHOT.FCS* as well.

```
SNAPSHOT_FILENAME   <file name>
```

Files added to the snapshots are said to be activated. Files may be activated whether or not the automatic snapshots are turned on in the configuration file. However, the activation has no effect until snapshots are written to the *SYSLOG* files.

The *<file name>* argument may include wildcard matching characters where “*” matches an arbitrary number of any characters and “?” matches exactly one of any character. A pattern of

simply "*" matches any user or file name. See **c-treeACE Standard Wildcards** (page 156). For example, the following keywords activate any file ending in ".dat", and the file *journal.idx*:

```
SNAPSHOT_FILENAME   *.dat
SNAPSHOT_FILENAME   journal.idx
```

File name case sensitivity depends on the platform. For example, Windows is case insensitive and Unix is case sensitive. The file names activated must match the file name used to first open the file. In particular, paths used in the activation list and during the call to open the file must match.

See Also

SNAPSHOT_INTERVAL (page 266)
SNAPSHOT_USERID (page 266)

SNAPSHOT_INTERVAL

The SNAPSHOT_INTERVAL keyword enables automatic snapshots at specified intervals:

```
SNAPSHOT_INTERVAL   <minutes>
```

By default, only the system snapshot is captured. To add user or file-specific snapshots to the data captured, use one or more of the following configuration entries:

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

See Also

SNAPSHOT_FILENAME (page 265)
SNAPSHOT_USERID (page 266)

SNAPSHOT_LOCKWAIT_USEC

```
SNAPSHOT_LOCKWAIT_USEC <lock wait histogram interval width in microseconds>
```

Histograms of waiting times for blocked lock requests are available: one for waiting times for blocked data record lock requests, and one for waiting times for blocked index lock requests (please note that the index locks are not controlled by the user). There is a small amount of overhead associated with mutex calls to collect clean statistics.

SNAPSHOT_LOCKWAIT_USEC can change the default histogram intervals (box width) for the lock waiting time histograms (default of 10,000 µsec or 0.01 seconds).

SNAPSHOT_TRANTIME_USEC

```
SNAPSHOT_TRANTIME_USEC <tran time histogram interval width in microseconds>
```

The *SNAPSHOT* feature includes a histogram of transaction times. There is a small amount of overhead associated with mutex calls to collect clean statistics.

SNAPSHOT_TRANTIME_USEC can change the default histogram intervals (box width) for the transaction time histogram (default of 50,000 µsec or 0.05 seconds).

SNAPSHOT_USERID

```
SNAPSHOT_USERID     <user ID>
```

By default, only the system snapshot is captured with `SNAPSHOT_INTERVAL`. `SNAPSHOT_USERID` is used to capture user information to *SNAPSHOT.FCS* as well.

Users added to the snapshots are said to be activated. Users may be activated whether or not the automatic snapshots are turned on in the configuration file. However, the activation has no effect until snapshots are written to the *SYSLOG* files.

The *<user ID>* argument may include wildcard matching characters: "*" matches an arbitrary number of any characters, and "?" matches exactly one of any character. A pattern of simply "*" matches any user or file name. See **c-treeACE Standard Wildcards** (page 156). For example, the following keywords activate all users:

```
SNAPSHOT_USERID     *
```

User IDs are not case sensitive.

See Also

```
SNAPSHOT_INTERVAL (page 266)
SNAPSHOT_FILENAME (page 265)
```

SYSLOG

```
SYSLOG <option>
```

The c-treeACE Server maintains two system files, *SYSLOGDT.FCS* and *SYSLOGIX.FCS*, for recording system events. Unlike the *CTSTATUS.FCS* file, the system log files can be encrypted such that entries cannot be added, deleted, or modified with a simple text editor, and vendors can add application-specific entries to the log.

The System Event Log contents are entered as pairs in the form of: `SYSLOG <keyword>`. As many of these pairs as desired may be used at the direction of your vendor. Current `SYSLOG` options include:

```
ADMIN_API
CTSTATUS
DELETE_FILE
DISABLE_API
DYNAMIC_DUMP
ENCRYPT
USER_INFO
NONE
LOGFAIL_PURGE
LOGFAIL_CTSTATUS
LOGFAIL_TERMINATE
```

Refer to the Server System Event Log (page 138) section for complete details of all *SYSLOG* feature options available.

Note: There is currently a 4GB limit to the SYSLOG files. It is highly recommended to limit information recorded in this file in high volume systems, and include the `LOGFAIL_PURGE` option to clear data as the file grows.

SYSVIEW_WHAT

```
SYSVIEW_WHAT <info>
```

This option supports the system status reports triggered by specified events such as increasing the number of log files or beginning a dynamic dump. *<info>* is one of the following to determine what to include in the report.

```
USER_COUNT
USER_LIST
FILES_COUNT
FILES_LIST
TRANS_COUNT
TRANS_LIST
MEMORY
LOGS
ALL
```

One or more configuration entries can be specified in *ctsrvr.cfg*. If the ALL parameter is used, any other entry is redundant. The LIST options (for example., USER_LIST) also report the simple COUNT statistics (for example., USER_COUNT) such that it is redundant to specify both a LIST and COUNT. The only reason to use the COUNT option is to reduce output.

The reports are added to the end of the *SYSVIEW.FCS* text file. Each report is time stamped and the triggering event is noted.

Example

```
SYSVIEW_WHEN   LOG_ACTIVE_CHG
SYSVIEW_WHEN   DYNDMP_BEG
SYSVIEW_WHAT   ALL
```

See Also

- SYSVIEW_WHEN (page 268)

Note: This sever feature is not currently enabled.

SYSVIEW_WHEN

```
SYSVIEW_WHEN  <event>
```

This option supports the system status reports triggered by specified events such as increasing the number of log files or beginning a dynamic dump. *<event>* is one of the following to determine when the report is generated:

```
CHECKPOINT_BEG
CHECKPOINT_END
SHUTDOWN
LOG_ACTIVE_CHG,
LOG_SIZE_CHG
DYNDMP_BEG
DYNDMP_END
ALL
```

The reports are added to the end of the *SYSVIEW.FCS* text file. Each report is time stamped and the triggering event is noted.

Example

```
SYSVIEW_WHEN   LOG_ACTIVE_CHG
SYSVIEW_WHEN   DYNDMP_BEG
SYSVIEW_WHAT   ALL
```

See Also

- SYSVIEW_WHAT (page 267)

Note: This sever feature is not currently enabled.

DIAGNOSTICS SNAPSHOT_AUTOMATIC

Write system snapshots to the human-readable *SNAPSHOT.FCS* text file.

```
DIAGNOSTICS    SNAPSHOT_AUTOMATIC
```

writes any automatic snapshots to *SNAPSHOT.FCS* instead of to the *SYSLOG* files. However, only the system snapshot is written. Snapshots for activated users and/or files are ignored.

DIAGNOSTICS SNAPSHOT_IOTIME

```
DIAGNOSTICS SNAPSHOT_IOTIME
```

Enables collecting disk read and write timings on a per-file basis when high-resolution timer support is activated.

DIAGNOSTICS SNAPSHOT_SHUTDOWN

Write system snapshots to the human-readable *SNAPSHOT.FCS* text file.

```
DIAGNOSTICS    SNAPSHOT_SHUTDOWN
```

Writes a system snapshot to *SNAPSHOT.FCS* at the start of the server shutdown process. However, only the system snapshot is written. Snapshots for activated users and/or files are ignored.

FairCom recommends this option in all configuration files to establish baseline statistics over time.

DIAGNOSTICS SNAPSHOT_WORKTIME

```
DIAGNOSTICS SNAPSHOT_WORKTIME
```

Enables collecting c-treeACE function call counts and timings on a per-c-tree file basis.

11.11 Security Keywords

c-treeACE provides a variety of keywords that can be used for security purposes.

Encryption

ADMIN_ENCRYPT (page 272)

Encrypt the *FAIRCOM.FCS* file at the time it is created.

ADVANCED_ENCRYPTION (page 272)

Enable advanced encryption for files.

LOG_ENCRYPT (page 275)

Camouflages the contents of the transaction logs to prevent unauthorized access.

MASTER_KEY_FILE (page 276)

Specifies a file from which c-tree reads the master encryption key.

User Access

LOGON_FAIL_LIMIT (page 275)

Specifies the optional limit on the number of consecutive failed logons that causes subsequent logon attempts to fail for `LOGON_FAIL_TIME` minutes.

LOGON_FAIL_TIME (page 276)

The length of time logons are blocked after the logon limit is exceeded.

LOGON_MUST_TIME (page 276)

Requires users to log on “at-least-once” within the specified time.

STARTUP_BLOCK_LOGONS (page 277)

Prevents non-ADMIN user logons when the server is started.

Tamper-Proof Settings

These keywords affect people's ability to alter system integrity by overriding settings from a command line or by altering configuration files. See also *Settings File* (page 158).

NULL_STRING (page 277)

Defines a symbol that represents a null string so that options can be blocked in the settings file without activating them.

COMPATIBILITY NO_COMMAND_LINE (page 273)

Instructs c-treeACE to ignore command-line arguments.

COMPATIBILITY NO_CONFIG_FILE (page 273)

Instructs c-treeACE to ignore the standard configuration file, *ctsrvr.cfg*.

Restrictions

ENABLE_TRANSFER_FILE_API (page 274)

Enables the file transfer function, **ctTransferFile()**, which is used to transfer a file to or from the server.

FILEDEF_SECURITY_LEVEL (page 275, **http://docs.faircom.com/doc/ctserver/#57477.htm**)

Protects the resource APIs, **ADDRES()**, **UPDRES()**, and **DELRES()**, with safeguards against unauthorized modification of file definition resources such as *IFIL* definitions, conditional indices, row-level filters, etc.

Security-Related Compatibility Options

COMPATIBILITY NONADMIN_FILBLK (page 273)

Permits a non-ADMIN user to set a file block if the blocking user has the file opened with update permissions.

COMPATIBILITY NONADMIN_QUIET (page 274)

Permits a non-ADMIN user to call **ctQuiet()** to quiesce the server.

COMPATIBILITY NONADMIN_TRANSFER_FILE (page 274)

Permits a non-ADMIN user to call **ctTransferFile()** to transfer a file.

COMPATIBILITY NON_ADMIN_SHUTDOWN (page 274)

Allows non-ADMIN users to shut down the Server.

ADMIN_ENCRYPT

```
ADMIN_ENCRYPT <YES | NO>
```

Encrypt the *FAIRCOM.FCS* file at the time it is created. **Note**: This keyword will not encrypt an *existing* file.

To enable the encryption of the user group information file, *FAIRCOM.FCS*, set `ADMIN_ENCRYPT YES` (default). The default setting uses AES encryption, preventing casual inspection of the data on disk, adding an additional level of security for passwords and user definitions stored in the file.

To disable the encryption of this file, set `ADMIN_ENCRYPT No`.

> **Note**: AES encryption is only available if `ADVANCED_ENCRYPTION YES` is specified in *ctsrvr.cfg*. This implies that, by default, the `ADMIN_ENCRYPT` option encrypts *FAIRCOM.FCS* using FairCom's "camo" encryption.

Default: YES

ADVANCED_ENCRYPTION

```
ADVANCED_ENCRYPTION < YES | NO >
```

Enable advanced encryption for files.

When Advanced Encryption is enabled, c-treeACE prompts for a master password at server startup. Run the **ctcpvf** utility to generate an encrypted password for use when launching the Advanced Encryption enabled Server. This will generate the file *ctsrvr.pvf*.

> **Note:** Developers can use the c-treeACE SDK to replace this prompt with an application-specific method of retrieving the master password.

Any time you change the advanced encryption setting, you should delete the *FAIRCOM.FCS* file (which contains user and group information) before restarting c-treeACE as user and group information is encrypted for protection as well. All user and group information must be recreated if the *FAIRCOM.FCS* file is deleted.

Client implementation of Advanced Encryption is accomplished through the use of the **SetEncryption()** function. Refer to the *c-tree Plus Function Reference Guide* for details on this function.

Default: NO

See Also

- MASTER_KEY_FILE (page 276)

ALLOW_MASTER_KEY_CHANGE

c-treeACE supports a master key file in which the advanced encryption master key can be stored. The `MASTER_KEY_FILE` configuration option is used to specify a local encrypted key store.

To prevent tampering with advanced encryption keys, the server configuration option `ALLOW_MASTER_KEY_CHANGE` must be specified to allow changing master passwords via the **SECURITY** API function (an administrative client-side API call). Default value: NO.

Note: The server configuration option `MASTER_KEY_FILE` now considers a value of NONE to indicate no master key file is in use. This option can be specified in a settings file to prevent a configuration file from using an existing master key file.

COMPATIBILITY NO_COMMAND_LINE

`COMPATIBILITY NO_COMMAND_LINE`

This option is used in conjunction with the tamper-proof settings file under the server. Configuration options that are in the encrypted settings file, *ctsrvr.set*, cannot be overridden in the *ctsrvr.cfg* file. This keyword instructs the c-treeACE to ignore the command line arguments.

Default: Not present

See Also

- COMPATIBILITY NO_CONFIG_FILE (page 273)
- COMPATIBILITY NONE (page 324)
- DIAGNOSTICS NONE (page 354)

COMPATIBILITY NO_CONFIG_FILE

`COMPATIBILITY NO_CONFIG_FILE`

This option is used in conjunction with the tamper-proof settings file under the server. Configuration options that are in the encrypted *ctsrvr.set* settings file cannot be overridden in the *ctsrvr.cfg* file. This keyword instructs the c-treeACE to ignore the standard configuration file, *ctsrvr.cfg*.

Default: Not present

See Also

- COMPATIBILITY NO_COMMAND_LINE (page 273)
- COMPATIBILITY NONE (page 324)
- DIAGNOSTICS NONE (page 354)

COMPATIBILITY NONADMIN_FILBLK

`COMPATIBILITY NONADMIN_FILBLK`

Permits a non-ADMIN user to set a file block if the blocking user has the file opened with update permissions.

If a non-ADMIN user attempts to set a file block without this keyword and having update permissions on the file, then error **LADM_ERR** (589) will be returned.

COMPATIBILITY NONADMIN_QUIET

```
COMPATIBILITY NONADMIN_QUIET
```

Permits a non-ADMIN user to call **ctQuiet()** to quiesce the server.

If a non-ADMIN user attempts to call **ctQuiet()** without this keyword, then error **LADM_ERR** (589) will be returned.

COMPATIBILITY NONADMIN_TRANSFER_FILE

```
COMPATIBILITY NONADMIN_TRANSFER_FILE
```

The file transfer API function **ctTransferFile()** can be used to transfer a file to or from the server. By default, this function is disabled for security reasons and is restricted to a member of the ADMIN group. This configuration keyword permits a non-ADMIN user to call **ctTransferFile()** to transfer a file.

If a non-ADMIN user attempts to call **ctTransferFile()** without this keyword, then error **LADM_ERR** (589) will be returned.

See Also

`ENABLE_TRANSFER_FILE_API` (page 274)

COMPATIBILITY NON_ADMIN_SHUTDOWN

```
COMPATIBILITY NON_ADMIN_SHUTDOWN
```

Allow non-ADMIN users to shut down the Server.

Default: ADMIN group only may shutdown.

ENABLE_TRANSFER_FILE_API

```
ENABLE_TRANSFER_FILE_API < YES | NO >
```

The file transfer API function **ctTransferFile()** can be used to transfer a file to or from the server. By default, this function is disabled for security reasons and is restricted to a member of the ADMIN group. To enable this feature, this keyword must be specified to YES in the c-treeACE configuration file, *ctsrvr.cfg*. Calling this function without this keyword enabled will result in a **NSUP_ERR** (454, feature not supported) error returned to the client.

Default: NO

See Also

`COMPATIBILITY NONADMIN_TRANSFER_FILE` (page 274)

FILEDEF_SECURITY_LEVEL

```
FILEDEF_SECURITY_LEVEL    <level>
```

With row-level, system-wide, data filters, it is necessary to protect the resource APIs, **ADDRES()**, **UPDRES()**, **DELRES()**, with safeguards against unauthorized modification of file definition resources such as *IFIL* definitions, conditional indices, row-level filters, etc.

The c-tree Server can enforce different levels of File Definition Resource (FCRES) security when users modify data file definition resources such as the *IFIL* and *DODA* resources. The actual level of protection enforced for a given resource is determined by the `FILEDEF_SECURITY_LEVEL` configuration keyword.

The file definition resource security keyword must be specified in the c-tree Server configuration file *ctsrvr.cfg*. There are three different security levels for this keyword:

`FILEDEF_SECURITY_LEVEL LOW`

This is the lowest security setting. There is no protection as any user may add or delete data file definition resources. This setting may be used to keep the c-tree Server data compatible with legacy applications.

`FILEDEF_SECURITY_LEVEL MEDIUM`

This is the default security setting. Any user may add or delete data file definition resources, but the file must be opened exclusive. This default setting may be enough to keep the c-tree Server data compatible with most legacy applications.

`FILEDEF_SECURITY_LEVEL HIGH`

This is the highest security setting. A user must have file definition permission before a definition resource is added or deleted. The file must be opened exclusive. This setting is appropriate for applications that require the highest level of security and may cause compatibility problems with existing legacy applications.

Default: MEDIUM.

Note: This feature is NOT currently enabled.

LOG_ENCRYPT

```
LOG_ENCRYPT YES
```

`LOG_ENCRYPT YES` camouflages the contents of the transaction logs to prevent unauthorized access.

Default: NO

LOGON_FAIL_LIMIT

```
LOGON_FAIL_LIMIT <attempts>
```

The optional limit on the number of consecutive failed logons that causes subsequent logon attempts to fail for `LOGON_FAIL_TIME` minutes. A logon which fails during this period returns **LRSM_ERR** (584).

Default: 0 (no limit)

See Also

LOGON_FAIL_TIME (page 276)
LOGON_MUST_TIME (page 276)

LOGON_FAIL_TIME

```
LOGON_FAIL_TIME <minutes>
```

The length of time logons are blocked after the logon limit is exceeded. A value of -1 indicates that there should be a permanent “log-on block” placed on a user until the Server ADMIN intervenes.

Default: 5

See Also

LOGON_FAIL_LIMIT (page 275)
LOGON_MUST_TIME (page 276)

LOGON_MUST_TIME

```
LOGON_MUST_TIME <minutes>
```

A non-zero value requires users to log on “at-least-once” within the defined time (e.g: at least once a week). If the time expires for a specific user, their profile will be deactivated, preventing access to the c-treeACE Server. The Server Administrator, or other ADMIN group user, must re-set the user’s account once the time limit has elapsed.

Default: 0 (no limit)

See Also

LOGON_FAIL_LIMIT (page 275)
LOGON_FAIL_TIME (page 276)

MASTER_KEY_FILE

```
MASTER_KEY_FILE  <filename>
```

Specifies a file from which c-tree reads the master encryption key. On Linux 2.6 and later kernel systems, c-tree uses the keyutils support to create a user-specific key in which the master key is stored. On other Unix systems, the master key is stored in a file on disk, with permissions set so that only the user that created the file can read it (permissions are set to 400).

The file (or user key on Linux) is encrypted using AES, however, the encryption is intended to only prevent casual inspection of the data when the file's contents are viewed. The permissions on the file are the defense against an unauthorized user reading the file.

The **ctcpvf** utility's -s option is used to create the master key file.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

See Also

- ADVANCED_ENCRYPTION (page 272)

NULL_STRING

```
NULL_STRING null
```

This option is used in conjunction with the tamper-proof settings file under the server. Configuration options that are in the encrypted settings file, *ctsrvr.set*, cannot be overridden in the *ctsrvr.cfg* file. The `NULL_STRING` keyword lets you define a symbol that represents a null string so that options can be blocked in the settings file without activating them.

```
LOCAL_DIRECTORY null
```

With the above entry example in the server settings file, `LOCAL_DIRECTORY` is blocked from use in the standard *ctsrvr.cfg* file but it is not activated.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: Not enabled

STARTUP_BLOCK_LOGONS

```
STARTUP_BLOCK_LOGONS
```

Prevents non-ADMIN user logons when the server is started. Only users in the ADMIN group are allowed to logon.

This feature allows the server to start for ADMIN purposes before authorizing access by non-ADMIN users. This could include creating or adjusting files, adjusting security options, or any other operations that require a functioning Server but are more conveniently accomplished when users are not connected to the Server.

Default: Allow logons

11.12 Backup Keywords

DYNAMIC_DUMP_DEFER (page 278)

Allows an administrator to reduce the performance impact of the dynamic dump.

DYNAMIC_DUMP_DEFER_INTERVAL (page 279)

Specifies the number of 64 KB blocks that are written before the `DYNAMIC_DUMP_DEFER` sleep is performed.

PERMIT_NONTRAN_DUMP (page 279)

Improves backup performance by skipping non-transaction files and pre-image files during a dynamic dump.

PREIMAGE_DUMP (page 280)

Causes all *PREIMG* files, even those not in a dynamic dump, to be temporarily changed to *TRNLOG* files and all transactions to be changed from *PREIMG* mode to *TRNLOG* mode during the dump.

VSS_WRITER (page 280)

When enabled, c-treeACE loads the Volume Shadow Copy Service (VSS) writer DLL (*c-treeACEVSSWriter.dll*) and initializes the VSS writer when the server starts.

Diagnostics

Diagnostics keywords are intended to be used to help collect data to be analyzed for troubleshooting. The use of these keywords should be avoided in normal operation as they may cause additional overhead (e.g., logging).

DIAGNOSTICS DYNDUMP_LOG (page 282)

Generates dynamic dump status info, sending progress entries during each dynamic dump to *CTSTATUS.FCS.*

DIAGNOSTICS VSS_WRITER (page 282)

Enables VSS writer to log diagnostic messages to *CTSTATUS.FCS*.

DYNAMIC_DUMP_DEFER

```
DYNAMIC_DUMP_DEFER <milliseconds>
```

When a dynamic dump runs, the disk read and write operations of the backup process can slow the performance of other database operations. This option allows an administrator to reduce the performance impact of the dynamic dump.

milliseconds is the time that the dynamic dump thread will sleep after each write of a 64KB block of data to the dump backup file.

An application developer can also use the c-tree **ctSETCFG()** API function to set the *DYNAMIC_DUMP_DEFER* value. For example, the following call specifies a 10-millisecond DYNAMIC_DUMP_DEFER time:

- **ctSETCFG(** *setcfgDYNAMIC_DUMP_DEFER, "10"* **);**

The *DYNAMIC_DUMP_DEFER* value set by a call to **ctSETCFG()** takes effect immediately, so this API call can be used by administrators to adjust the speed of a running dynamic dump depending on the amount of other database activity.

Note: The maximum allowed *DYNAMIC_DUMP_DEFER* time is 5000 milliseconds, set at compile-time. If a value is specified that exceeds this limit, the *DYNAMIC_DUMP_DEFER* time is set to *DYNAMIC_DUMP_DEFER_MAX*.

The c-treeACE Administrator utility, **ctadmn**, was also updated to support the dump sleep time option to change this value at run time. The "Change Server Settings" menu is available from the main menu of the **ctadmn** utility.

See Also

- DYNAMIC_DUMP_DEFER_INTERVAL (page 279)

DYNAMIC_DUMP_DEFER_INTERVAL

```
DYNAMIC_DUMP_DEFER_INTERVAL <blocks>
```

<blocks> specifies the number of 64 KB blocks that are written before the `DYNAMIC_DUMP_DEFER` sleep is performed.

The `DYNAMIC_DUMP_DEFER` option causes the dynamic dump to pause for the specified number of milliseconds each time it writes 64 KB of data to the dynamic dump stream file. For large backups, even the smallest DYNAMIC_DUMP_DEFER value of 1 millisecond adds significant time to the dynamic dump. For example 100 GB = 1600000 * 1 ms. = 1600 seconds of additional time.

Note: If a value greater than 5000 is specified for `DYNAMIC_DUMP_DEFER_INTERVAL`, the value is set to 5000. If a value less than 1 is specified, the value is set to 1.

This option can be set by the **ctSETCFG()** API function. A new menu option to set this value has been added to option 10 of the c-treeACE Server Administration (**ctadmn**) menu.

Example

```
DYNAMIC_DUMP_DEFER_INTERVAL 16
```

Causes the `DYNAMIC_DUMP_DEFER` sleep to occur after every 64 KB * 16 = 1 MB of data written to the dump stream file.

PERMIT_NONTRAN_DUMP

```
PERMIT_NONTRAN_DUMP < YES | NO >
```

Improves backup performance by skipping non-transaction files and pre-image files during a dynamic dump.

Default: YES

When set to NO, non-transaction files and pre-image files are skipped during a dynamic dump even if they are included in the !FILES section of the dynamic dump configuration script. The exception is if PREIMAGE_DUMP is set to YES, then pre-image files continue to be included in the dump.

PREIMAGE_DUMP

```
PREIMAGE_DUMP <YES | NO>
```

When enabled, all *PREIMG* files, even those not in a dynamic dump, are temporarily changed to *TRNLOG* files to be compatible with the upgraded transactions, and all transactions are automatically changed from *PREIMG* mode to *TRNLOG* mode during the dump. *PREIMG* files opened or created in the middle of the dump are also temporarily promoted from *PREIMG* files to *TRNLOG* files. All promoted files are restored to their *PREIMG* status at the conclusion of the dynamic dump.

The dynamic dump script file accepts a *!DELAY* parameter whose argument is the number of seconds to wait for a transaction to complete before aborting it in order to permit the start of a dynamic dump. When the `PREIMAGE_DUMP` facility is used, the *!DELAY* parameter is effectively ignored if a long *PREIMG* transaction begins prior to the dynamic dump. This means the dump will not begin until all current transactions complete. See "Dynamic Dump" (page 98) and "Advanced - Faster Auto-Recovery" (page 98) for additional information.

Default: NO

See Also

AUTO_PREIMG (page 209)
AUTO_TRNLOG (page 209)
AUTO_TRNLOG_LIGHT (page 210)
DIAGNOSTICS AUTO_TRNLOG_CHECKLOCK (page 348)
DIAGNOSTICS AUTO_PREIMG_CHECKLOCK (page 348)
DIAGNOSTICS AUTO_TRNLOG_CHECKREAD (page 348)
DIAGNOSTICS AUTO_PREIMG_CHECKREAD (page 348)

VSS_WRITER

```
VSS_WRITER YES
```

With this option enabled, c-treeACE loads the Volume Shadow Copy Service (VSS) writer DLL (*c-treeACEVSSWriter.dll*) and initializes the VSS writer when the server starts.

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

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

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

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

Compatibility Notes

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

Files to Be Backed Up

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

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

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

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

See Also

`DIAGNOSTICS VSS_WRITER (page 282)`

DIAGNOSTICS DYNDUMP_LOG

DIAGNOSTICS DYNDUMP_LOG

The DIAGNOSTICS DYNDUMP_LOG keyword generates dynamic dump status info, sending progress entries during each dynamic dump to *CTSTATUS.FCS*, including an entry for each file it attempts to dump.

Default: OFF

DIAGNOSTICS VSS_WRITER

DIAGNOSTICS VSS_WRITER

The following c-treeACE configuration option enables VSS writer diagnostic logging:

```
DIAGNOSTICS VSS_WRITER
```

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

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

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

See Also

VSS_WRITER (page 280)

11.13 Replication Keywords

Data replication is a powerful c-treeACE feature providing near real-time data availability between one or more servers. These options configure specific attributes of this optional feature.

REPLICATE (page 285)

Specifies which files are required to be replicated via the Replication Agent.

REPL_IDENTITY_USE_MASTER (page 286)

Forces the local c-treeACE process to use the serial number value from the master server when adding a record to a local replica.

REPL_IDENTITY_USE_SOURCE (page 286)

Forces the target c-treeACE process to use the identity field value from the source server when adding a record to a replicated file.

REPL_MAPPINGS (page 286)

Specifies a local filename mapping to a master server.

REPL_NODEID (page 286)

Sets the replication node ID of a local c-tree Server.

REPL_READ_BUFFER_SIZE (page 287)

Sets the size of the replication log read buffer.

REPL_SRLSEG_ALLOW_UNQKEY (page 287)

Forces c-treeACE to allow an index that contains a *SRLSEG* segment to be the replication unique key.

REPL_SRLSEG_USE_MASTER (page 288)

Forces a local c-treeACE process to fill in the serial number value from the master c-treeACE process when adding a record to a local replica.

REPL_SRLSEG_USE_SOURCE (page 288)

Forces a c-treeACE process replication writer thread to fill in the serial number value from the source server when adding a record to a replicated file to preserve existing serial numbering.

Diagnostics

Diagnostics keywords are intended to be used to help collect data to be analyzed for troubleshooting. The use of these keywords should be avoided in normal operation as they may cause additional overhead (e.g., logging).

DIAGNOSTICS REPLICATE (page 288)

Enables c-treeACE to log messages to *CTSTATUS.FCS* when a file that is being created or opened matches the filename specified for the `REPLICATE` keyword, and it does not meet replication requirements.

Auto-Numbering Replication Defaults Changed

In V11, changes have been made to the c-treeACE Server defaults for replication behavior on *SRLSEG* and identity fields.

Compatibility Note: This is a change in behavior, but many of the c-treeACE Server configuration files included in FairCom products already contain this option so the behavior is already in effect in those cases. However, the c-treeRTG package did not include these options, so this will change the default behavior for c-treeRTG COBOL Edition and c-treeRTG BTRV Edition.

The default values have been changed for the following options:

REPL_SRLSEG_ALLOW_UNQKEY now defaults to YES

This option allows a *SRLSEG* index to be used as a replication unique index. Without this option, a data file whose only unique index is a *SRLSEG* index would not qualify for replication. By changing this default to YES, the *SRLSEG* index can be used as a replication unique index, which is the commonly-expected behavior.

REPL_SRLSEG_USE_SOURCE now defaults to YES

This option uses the serial number value from the source c-treeACE Server when adding a record to a replicated file. This option applies to c-tree's asynchronous (source/target) replication model, in which a Replication Agent replicates changes from a source c-treeACE Server to a target c-treeACE Server. By changing this default to YES, records added to the target server will contain the *SRLSEG* value from the record on the source server, rather than the target server generating its own *SRLSEG* value for the new record. This default option corresponds to the behavior that is most likely to be expected.

REPL_SRLSEG_USE_MASTER now defaults to YES

This option uses the serial number value from the master c-treeACE Server when adding a record to a replicated file. This option applies to c-tree's synchronous (local/master) replication model, in which an update to a record on a local c-treeACE Server triggers an update to the record in the associated table on the master c-treeACE Server. By changing this default to YES, records added to the local server will contain the *SRLSEG* value from the record on the master server, rather than the local server generating its own SRLSEG value for the new record. This default option corresponds to the behavior that is most likely to be expected.

REPL_IDENTITY_USE_SOURCE now defaults to YES

This option uses the identity field value from the source c-treeACE Server when adding a record to a replicated file. This option applies to c-tree's asynchronous (source/target) replication model, in which a Replication Agent replicates changes from a source c-treeACE Server to a target c-treeACE Server. By changing this default to YES, records added to the target server will contain the identity field value from the record on the source server, rather than the target server generating its own identity field value for the new record. This default option corresponds to the behavior that is most likely to be expected.

REPL_IDENTITY_USE_MASTER now defaults to YES

This option uses the identity field value from the master c-treeACE Server when adding a record to a replicated file. This option applies to c-tree's synchronous (local/master) replication model, in which an update to a record on a local c-treeACE Server triggers an update to the record in the associated table on the master c-treeACE Server. By changing this default to YES, records added to the local server will contain the identity field value from the record on the master server, rather than the local server generating its own identity field value for the new record. This default option corresponds to the behavior that is most likely to be expected.

REPLICATE

```
REPLICATE <filename>
```

The source replica only needs to specify which files are required to be replicated via the Replication Agent. The `REPLICATE` keyword

```
REPLICATE <filename>
```

is used to specify which file(s) to enable transaction log information for replication. Multiple `REPLICATE` statements are allowed. The file name may include wildcard characters (see **c-treeACE Standard Wildcards** (page 156)).

> **Note**: Choose your wild cards carefully because certain FairCom files, such as *REPLSTATEDT.FCS*, must not be replicated. For example, `REPLICATE *.dat` is preferable to `REPLICATE *`.

Individual files can also be enabled programmatically through the replication API call **ctReplSetFileStatus()**.

> **Note:** The low-level c-treeACE **UpdateHeader()** API can be used to enable replication on a per file basis. Use the *ctREPLCIATEhdr mode* with either NO (off) or YES (on) for the *hdrval*. If the file does not have an extended header, then the change will not persist once the file is closed and reopened. Because failure to enforce replication is a serious problem, **UpdateHeader()** returns a special error code, **-EXTH_ERR** (-734), when the request succeeds, but the file header is not extended. This is more of a warning than an error.

See Also

- REPL_READ_BUFFER_SIZE (page 287)
- REPL_MAPPINGS (page 286)

REPL_IDENTITY_USE_MASTER

```
REPL_IDENTITY_USE_MASTER  <YES | NO>
```

Forces the local c-treeACE process to use the serial number value from the master server when adding a record to a local replica.

See Also

- REPL_IDENTITY_USE_SOURCE (page 286)
- REPL_SRLSEG_ALLOW_UNQKEY (page 287)
- REPL_SRLSEG_USE_MASTER (page 288)
- REPL_SRLSEG_USE_SOURCE (page 288)

REPL_IDENTITY_USE_SOURCE

```
REPL_IDENTITY_USE_SOURCE  <YES | NO>
```

Forces the target c-treeACE process to use the identity field value from the source server when adding a record to a replicated file.

See Also

- REPL_IDENTITY_USE_MASTER (page 286)
- REPL_SRLSEG_ALLOW_UNQKEY (page 287)
- REPL_SRLSEG_USE_MASTER (page 288)
- REPL_SRLSEG_USE_SOURCE (page 288)

REPL_MAPPINGS

```
REPL_MAPPINGS <mapfilename>
```

Specifies a local filename mapping to a master server. Multiple instances of this keyword can be used.

This keyword also forces the local c-treeACE process to attempt to load the multi-threaded c-tree client library (*mtclient.dll* or *libmtclient.so*), which it uses to communicate with a master c-treeACE process. If this library cannot be loaded, or if it does not contain the required functions, c-treeACE fails to start with the error **TR_CLIL_ERR** (850, Transactional replication: Failed to start c-tree remote client subsystem: see *CTSTATUS.FCS* for details).

See Also

- REPLICATE (page 285)
- REPL_READ_BUFFER_SIZE (page 287)

REPL_NODEID

```
REPL_NODEID  <nodeid>
```

c-treeACE assigns a node ID to a Replication Agent. The local/master replication scheme also assigns a node ID to connections that are established from the local server to the master server.

A local c-tree Server can set its replication node ID by using the `REPL_NODEID` option in *ctsrvr.cfg*. For example, consider a master server and two local servers:

```
; master server configuration
SERVER_NAME MASTER
REPL_NODEID 10.0.0.1

; local server 1 configuration
SERVER_NAME LOCAL01
REPL_NODEID 10.0.0.2

; local server 2 configuration
SERVER_NAME LOCAL02
REPL_NODEID 10.0.0.3
```

ID values are arbitrary and do not need to match the IP address of the system on which the c-tree Server is running. They only need to be unique and not change during replication. (This example demonstrates IP v4 addresses.)

The Replication Agent reads the node ID of the source and target servers. If the node ID is not set for a source or target server, the Replication Agent uses the IP address of the system on which that c-treeACE is running.

Note: `REPL_NODEID` should be used if the entry for `source_server` or `target_server` in *ctreplagent.cfg* does not specify an IP address (e.g., if localhost or the DNS name is used).

REPL_READ_BUFFER_SIZE

```
REPL_READ_BUFFER_SIZE  <size>
```

Sets the size of the replication log read buffer. Specify a value of zero to disable the use of the replication log read buffer.

Default: 8K

See Also

- REPLICATE (page 285)
- REPL_MAPPINGS (page 286)

REPL_SRLSEG_ALLOW_UNQKEY

```
REPL_SRLSEG_ALLOW_UNQKEY <YES | NO>
```

Forces c-treeACE to allow an index that contains a *SRLSEG* segment to be the replication unique key.

See Also

- REPL_SRLSEG_USE_MASTER (page 288)

- REPL_SRLSEG_USE_SOURCE (page 288)

REPL_SRLSEG_USE_MASTER

```
REPL_SRLSEG_USE_MASTER  <YES | NO>
```

Forces a local c-treeACE process to fill in the serial number value from the master c-treeACE process when adding a record to a local replica. (Synchronous master/local replication model.)

See Also

- REPL_SRLSEG_ALLOW_UNQKEY (page 287)
- REPL_SRLSEG_USE_SOURCE (page 288)

REPL_SRLSEG_USE_SOURCE

```
REPL_SRLSEG_USE_SOURCE <YES | NO>
```

Forces a c-treeACE process replication writer thread to fill in the serial number value from the source server when adding a record to a replicated file. (Standard replication model.) This preserves existing serial numbering.

See Also

- REPL_SRLSEG_ALLOW_UNQKEY (page 287)
- REPL_SRLSEG_USE_MASTER (page 288)

DIAGNOSTICS REPLICATE

```
DIAGNOSTICS REPLICATE
```

The configuration option `DIAGNOSTICS REPLICATE` enables c-treeACE to log messages to *CTSTATUS.FCS* when a file that is being created or opened matches the filename specified for the `REPLICATE` keyword, and it does not meet replication requirements. This is useful in initial setup to determine unexpected replication failures.

Sample Output

```
Mon Apr 09 10:48:21 2012
 - User# 00014  REPLICATE_DIAG: None of the indices for the file mark.dat qualify as a replication
unique key:
Mon Apr 09 10:48:21 2012
 - User# 00014  REPLICATE_DIAG: index 1 is not unique
Mon Apr 09 10:48:21 2012
 - User# 00014  REPLICATE_DIAG: index 1 supports null keys
Mon Apr 09 10:48:21 2012
 - User# 00014  REPLICATE_DIAG: index 2 is not unique
Mon Apr 09 10:48:21 2012
 - User# 00014  REPLICATE_DIAG: index 3 is not unique
Mon Apr 09 10:48:21 2012
 - User# 00014  REPLICATE_DIAG: index 3 supports null keys
Mon Apr 09 10:48:21 2012
```

```
- User# 00014  REPLICATE_DIAG: index 3 contains a SRLSEG segment
```

The **SetSystemConfigurationOption()** API function can be used to turn this diagnostic option on and off at run time.

11.14 Unicode Keywords

c-treeACE Unicode support requires a custom server build enabling this feature. Please contact your nearest FairCom office for current availability options.

ICU_LOCALE (page 290)

Specifies a language (ISO-639), a country (ISO-3166), and any system-dependent options.

ICU_OPTION (page 291)

Specifies an option.

LANGUAGE (page 291)

Support the r-tree language feature in an r-tree enabled server.

XTDKSEG_SEG_TYPE (page 292)

Specifies a server default for each type of extended segment definition supported.

XTDKSEG_SRC_TYPE (page 292)

Specifies a type.

XTDKSEG_SRC_SIZE (page 292)

Specifies a size.

XTDKSEG_FAILED_DEFAULT_OK (page 293)

Specifies if the server can begin if server default encounters an error.

ICU_LOCALE

```
ICU_LOCALE <locale string: xx_YY_Variant>
```

Where “xx” is the language as specified by ISO-639 (e.g., “fr” for French); ‘Y’ is a country as specified by ISO-3166 (e.g., “fr_CA” for French language in Canada); and the “Variant” portion represents system dependent options.

See Also

ICU_OPTION (page 291), LANGUAGE (page 291), XTDKSEG_SEG_TYPE (page 292), XTDKSEG_SRC_TYPE (page 292), XTDKSEG_SRC_SIZE (page 292), XTDKSEG_FAILED_DEFAULT_OK (page 293)

ICU_OPTION

```
ICU_OPTION  <option>
```

Where `<option>` is one of the following:

```
STRENGTH_PRIMARY
STRENGTH_SECONDARY
STRENGTH_TERTIARY
STRENGTH_QUATERNARY
STRENGTH_IDENTICAL
STRENGTH_DEFAULT
NORM_NONE
NORM_CAN_DECMP
NORM_CMP_DECMP
NORM_CAN_DECMP_CMP
NORM_CMP_DECMP_CAN
NORM_DEFAULT
LOCALE_SYSDEFAULT_NOTOK
LOCALE_FALLBACK_NOTOK
ATTR_FRENCH_ON
ATTR_FRENCH_OFF
ATTR_CASE_ON
ATTR_CASE_OFF
ATTR_DECOMP_ON
ATTR_DECOMP_OFF
ATTR_SHIFTED
ATTR_NONIGNR
ATTR_LOWER
ATTR_UPPER
ATTR_HANGUL
```

A configuration file may contain many `ICU_OPTION` entries. Some combinations of entries do not make sense and the behavior is not guaranteed if they are combined. For instance, using both of these entries is inappropriate:

```
ICU_OPTION              ATTR_LOWER
ICU_OPTION              ATTR_UPPER
```

See Also

ICU_LOCALE (page 290), LANGUAGE (page 291), XTDKSEG_SEG_TYPE (page 292), XTDKSEG_SRC_TYPE (page 292), XTDKSEG_SRC_SIZE (page 292), XTDKSEG_FAILED_DEFAULT_OK (page 293)

LANGUAGE

```
LANGUAGE
```

Support the r-tree language feature in an r-tree enabled server. The following arguments are supported:

```
ENGLISH
ITALIAN
SPANISH
PORTUGUES
SJIS
EUC
```

If more than one entry exists in the configuration file, the last occurrence is used.

Note: Refer to the Unicode documentation for complete details regarding this support.

See Also

ICU_LOCALE (page 290), ICU_OPTION (page 291), XTDKSEG_SEG_TYPE (page 292), XTDKSEG_SRC_TYPE (page 292), XTDKSEG_SRC_SIZE (page 292), XTDKSEG_FAILED_DEFAULT_OK (page 293)

XTDKSEG_SEG_TYPE

```
XTDKSEG_SEG_TYPE UNICODE_ICU
```

Specifies a server default for each type of extended segment definition supported.

Example

```
XTDKSEG_SEG_TYPE         UNICODE_ICU
ICU_LOCALE               "ar"
XTDKSEG_SRC_SIZE         12
XTDKSEG_SRC_TYPE         UTF8
ICU_OPTION               STRENGTH_TERTIARY
ICU_OPTION               NORM_DEFAULT
```

See Also

ICU_LOCALE (page 290), ICU_OPTION (page 291), LANGUAGE (page 291), XTDKSEG_SRC_TYPE (page 292), XTDKSEG_SRC_SIZE (page 292), XTDKSEG_FAILED_DEFAULT_OK (page 293)

XTDKSEG_SRC_TYPE

```
XTDKSEG_SRC_TYPE <type>
```

Where *<type>* is one of:

```
PROVIDED
UTF8
UTF16
```

See Also

ICU_LOCALE (page 290), ICU_OPTION (page 291), LANGUAGE (page 291), XTDKSEG_SEG_TYPE (page 292), XTDKSEG_SRC_SIZE (page 292), XTDKSEG_FAILED_DEFAULT_OK (page 293)

XTDKSEG_SRC_SIZE

```
XTDKSEG_SRC_SIZE <size>
```

Where *<size>* is one of:

```
PROVIDED
```

```
COMPUTED
<numeric value>
```

See Also

ICU_LOCALE (page 290), ICU_OPTION (page 291), LANGUAGE (page 291), XTDKSEG_SEG_TYPE (page 292), XTDKSEG_SRC_TYPE (page 292), XTDKSEG_FAILED_DEFAULT_OK (page 293)

XTDKSEG_FAILED_DEFAULT_OK

```
XTDKSEG_FAILED_DEFAULT_OK  <YES | NO>
```

- `YES` - Server can still begin if server default encounters an error.
- `NO` - Server cannot continue on error, which is the default behavior.

See Also

ICU_LOCALE (page 290), ICU_OPTION (page 291), LANGUAGE (page 291), XTDKSEG_SEG_TYPE (page 292), XTDKSEG_SRC_TYPE (page 292), XTDKSEG_SRC_SIZE (page 292)

11.15 Mirroring Keywords

Mirroring duplicates c-treeACE data between two specified volumes as a legacy backup strategy. These options control various aspects of mirroring for multiple types of c-tree files.

ADMIN_MIRROR (page 294)

Sets the location where *FAIRCOM.FCS* is mirrored.

LOG_EVEN_MIRROR (page 295)

Specifies an alternative path and name for even-numbered secondary transaction log files.

LOG_ODD_MIRROR (page 295)

Specifies an alternative path and name for odd-numbered secondary transaction log files.

MIRROR_DIRECTORY (page 295)

Permits mirrored files WITHOUT AN ABSOLUTE PATH NAME to be placed in a specified mirror directory.

MIRRORS (page 296)

Turns off all mirroring when or allows ordinary operation in which the filename determines whether or not file mirroring is in effect on a file-by-file basis.

SKIP_MISSING_LOG_MIRRORS (page 296)

Specifies if the c-treeACE Server should NOT terminate automatic recovery if it cannot find a log mirror to be recovered along with the primary file.

SKIP_MISSING_MIRRORS (page 296)

Specifies if the c-treeACE Server should NOT terminate automatic recovery if it cannot find a mirror to be recovered along with the primary file.

START_EVEN_MIRROR (page 297)

Specifies the alternative name for even numbered secondary start file.

START_ODD_MIRROR (page 297)

Specifies the alternative name for odd numbered secondary start file.

ADMIN_MIRROR

```
ADMIN_MIRROR <mirror_path\FAIRCOM.FCS>
```

Sets the location where *FAIRCOM.FCS* is mirrored.

Permits *FAIRCOM.FCS* to be mirrored. For example, where mirror_path is the path to the secondary storage location for *FAIRCOM.FCS*.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: No log mirror

LOG_EVEN_MIRROR

```
LOG_EVEN_MIRROR <full_path>L
```

The alternative name for even numbered secondary transaction log files. This keyword allows the even numbered secondary transaction log files to be mirrored to a location other than the primary transaction log files. This name must be in the form of an optional directory path and the single character 'L' (e.g., *E:\LOG2EVEN\L*). The transaction management logic automatically appends a seven-digit even number and the extension *.FCS* to the path provided.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: No log mirrors

See Also

LOG_ODD_MIRROR (page 295)
MIRRORS (page 296)
MIRROR_DIRECTORY (page 295)

LOG_ODD_MIRROR

```
LOG_ODD_MIRROR <full_path>L
```

The alternative name for odd numbered secondary transaction log files. This keyword allows the odd numbered secondary transaction log files to be mirrored to a different location than the primary transaction log files. This name must be in the form of an optional directory path and the single character 'L' (e.g., *F:\LOG2ODD\L*). The transaction management logic automatically appends a seven-digit even number and the extension *.FCS* to the path provided.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: No log mirrors

See Also

LOG_EVEN_MIRROR (page 295)
MIRRORS (page 296)
MIRROR_DIRECTORY (page 295)

MIRROR_DIRECTORY

```
MIRROR_DIRECTORY <directory name>
```

Permits mirrored files WITHOUT AN ABSOLUTE PATH NAME to be placed in a specified mirror directory. This is analogous to `LOCAL_DIRECTORY` except that it only applies to the mirror in a primary|mirror pair.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: Server directory or `LOCAL_DIRECTORY`

See Also

`MIRRORS (page 296)`
`LOG_EVEN_MIRROR (page 295)`
`LOG_ODD_MIRROR (page 295)`

MIRRORS

`MIRRORS <YES | NO>`

Turns off all mirroring when set to NO. YES implies ordinary operation in which the filename determines whether or not file mirroring is in effect on a file-by-file basis. NO implies all mirror requests are ignored (including log files, administrative files, and all user mirrors). Set `MIRRORS` to NO only if there are strictly no plans to ever use file mirroring or during catastrophe recovery situations where the mirrored files may not be available due to a hardware problem. The absence of this keyword implies file mirrors are supported.

Logging of the message `Following file is opened without mirror...` to *CTSTATUS.FCS* is suppressed when the configuration file specifies `MIRRORS NO`, which disables the use of mirrors.

Default: YES

See Also

`MIRROR_DIRECTORY (page 295)`
`LOG_EVEN_MIRROR (page 295)`
`LOG_ODD_MIRROR (page 295)`

SKIP_MISSING_LOG_MIRRORS

`SKIP_MISSING_LOG_MIRRORS <YES | NO>`

Accepts a YES/NO argument. With an argument of YES, the c-treeACE Server does NOT terminate automatic recovery if it cannot find a log mirror to be recovered along with the primary file.

Default: NO

See Also

- SKIP_MISSING_FILES (page 232)
- SKIP_MISSING_MIRRORS (page 296)
- SKIP_INACCESSIBLE_FILES (page 232)

SKIP_MISSING_MIRRORS

`SKIP_MISSING_MIRRORS <YES | NO>`

Accepts a YES/NO argument. With an argument of YES, the c-treeACE Server does NOT terminate automatic recovery if it cannot find a mirror to be recovered along with the primary file.

Default: NO

See Also

- SKIP_MISSING_FILES (page 232)
- SKIP_MISSING_LOG_MIRRORS (page 296)
- SKIP_INACCESSIBLE_FILES (page 232)

START_EVEN_MIRROR

```
START_EVEN_MIRRORS <full_path>S
```

The alternative name for even numbered secondary start file. The start file contains the location at which the automatic recovery routines begin to scan the transaction logs. There are two start files (numbered zero and one) to reduce the risk of losing the starting point for automatic recovery. This name must be in the form of a directory path and the single character 'S' (e.g., *C:\START\S*). The c-treeACE Server appends a seven-digit even number and the extension *.FCS* to the name provided.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: No start mirrors

See Also

START_EVEN (page 223)
START_ODD (page 224)
START_ODD_MIRROR (page 297)

START_ODD_MIRROR

```
START_ODD_MIRRORS <full_path>S
```

The alternative name for odd numbered secondary start file. The start file contains the location at which the automatic recovery routines begin to scan the transaction logs. There are two start files (numbered zero and one) to reduce the risk of losing the starting point for automatic recovery. This name must be in the form of a directory path and the single character 'S' (e.g., *C:\START\S*). The c-treeACE Server appends a seven digit odd number and the extension *.FCS* to the name provided.

In V10.3 and later, this configuration option can include an environment variable name that will be substituted with its value when the configuration file is read.

Default: No start mirrors

See Also

START_EVEN (page 223)
START_EVEN_MIRROR (page 297)
START_ODD (page 224)

11.16 Backward Compatibility Keywords

Compatibility keywords are frequently used to provide extended "non-standard" c-treeACE behavior or backward compatibility with previous behaviors as server functionalities are changed. Typically, these keywords should only be used under the specific advice of a FairCom engineer as they may negatively impact the functionality of your application. Frequently, these keywords are used as the result of a required change in server behavior resulting in incompatibility with legacy applications. Please don't hesitate to contact your nearest FairCom office should you have any questions regarding use of any of these keywords.

WARNING: The keywords in this section should be used ONLY on the advice of your application developer. They can seriously alter the operation of the c-treeACE Server.

COMPATIBILITY 6BTRAN_NOT_DEFAULT (page 305)

Affects six-byte transaction number support.

COMPATIBILITY ABORT_ON_CLOSE (page 305)

Disable “deferred close” capability.

COMPATIBILITY BATCH_SIGNAL (page 305)

Causes *sysiocod* to be set to **BTNO_COD** (-844) on calls to **BATSETX()**/**BATSET()** that result in the current batch being closed.

COMPATIBILITY BATCH_UTRFMKEY (page 306)

Reverts to this prior behavior of the order that keys returned with **BATSET()** called with *BAT_RET_KEY*

COMPATIBILITY BLOCK_DDSFM_CREATE and **BLOCK_DDSFM_DELETE** (page 306)

Blocks superfile member creation and deletion during a dynamic dump.

COMPATIBILITY CHAR_SCHSEG (page 306)

For compatibility with versions prior to V8, reverts the behavior of treating *CT_CHAR* fields as signed, one-byte integers.

COMPATIBILITY CHECKPOINT_OVERLAP (page 306)

Reverts a change to the checkpoint logic causes checkpoint requests to wait for the completion of an in-process checkpoint before beginning even flushing operations.

COMPATIBILITY CHKPNT_FLUSHALL (page 307)

Forces all checkpoints but the final checkpoint to use a single system call to flush file system buffers instead of individual calls for each file to **ctsave()**.

COMPATIBILITY CHKPNT_MUTEX_REL (page 307)

Forces transaction related mutexes to be released just before the checkpoint is written to disk instead of after. For FairCom internal use only.

COMPATIBILITY CLSFIL_ISAM (page 307)

Reverts to the behavior prior to V10 in which low-level commands were allowed to close files opened with ISAM-level opens.

COMPATIBILITY CLSFIL_UNBLOCK (page 307)

Allows the file blocker to have the file open at the time of the unblock call.

COMPATIBILITY COMMPORT5000 (page 307)

Sets the default base value to add to the `SERVER_NAME` ASCII sum for determining the TCP/IP port number.

COMPATIBILITY CTREE_RWLOCK (page 308)

Forces c-treeACE to use custom reader/writer lock support instead of Windows' reader/writer lock support even if the Windows system supports reader/writer locks.

COMPATIBILITY DIR_BUF_RQS (page 308)

Disables changes that (a) avoid retries when the node holds the desired buffer and (b) skip the queue and directly request the holding semaphore. For internal FairCom testing only.

COMPATIBILITY DUPL_ERR_FATAL (page 308)

Forces the index corrupt flag to be set for a **DUPJ_ERR**.

COMPATIBILITY ENCRYPT128 (page 309)

Reverts to the original 128-bit master key usage for encryption.

COMPATIBILITY ESTIMATE_SCALE (page 309)

Reverts to prior behavior in which **ESTKEY()** returned the estimate to within 1/10 percent.

COMPATIBILITY EXACT_FILE_NAMES (page 309)

Forces different references to the same file to use the same names.

COMPATIBILITY EXTENDED_TRAN_ONLY (page 309)

Forces a **R6BT_ERR** (745) on an attempt to create or open a non-extended-transaction-number file.

COMPATIBILITY FAILED_TRAN_IO (page 310)

Restores behavior of a failed write operation for transaction-controlled files terminating the server (or stand-alone application) with error **L60** (*ctcatend*).

COMPATIBILITY FILE_CREATE_CHECKPOINT (page 310)

Restores the old behavior of generating a checkpoint if a TRNLOG (but not TRANDEP) file is about to be deleted or renamed after a file, any file, is created but with no intervening checkpoint.

COMPATIBILITY FILE_DESCRIPTOR_LIMIT (page 310)

Overrides the behavior described in FILES in case it is not convenient for a system administrator to set the file descriptor limit for the c-treeACE Server process to the required value or it is not desired to decrease the FILES or CONNECTIONS settings. This keyword is provided for backward compatibility or short-term use only - Other use is not recommended.

COMPATIBILITY FILELIST_GROWTH (page 311)

Reverts a change that increased the rate at which an array grows by doubling in size up to 2 thousand files.

COMPATIBILITY LFL_WAIT (page 311)

Eliminates the lflsema SHORTWAIT loop, and uses a call with the WAIT parameter. For internal FairCom use only.

COMPATIBILITY LFW_ADAPTIVE (page 311)

Replaces the commit delay **ct_udefer()** call with a loop that permits the "prime" cohort to periodically wake up, after a much smaller defer time, and check for commit delay progress. For internal FairCom use only.

COMPATIBILITY LOCK_CACHE (page 311)

Allows c-tree attempts to lock all data and index cache pages into physical memory, using the **mlock()** function on Unix systems. Currently disabled.

COMPATIBILITY LOCK_HEADER (page 312)

Restores the use of lock table entries for header calls that acquire the header semaphore.

COMPATIBILITY LOG_ENCRYPT128 (page 312)

Reverts to the original 128-bit master key usage for log encryption.

COMPATIBILITY PUTHDR_COMMIT (page 312)

Restores the original **PUTHDR()** behavior, which was changed such that on commit, the header value is not reset to the new value.

COMPATIBILITY MEMORY_FILE_SKIP_FREE (page 313)

Skips the free of memory at server shutdown for the individual memory records for memory files still open and assume that the server termination will (automatically) return the memory to the system.

COMPATIBILITY MEMORY_LIMITS (page 313)

Legacy V7 backward compatibility option that enforces system or user memory limits.

COMPATIBILITY MULTI_PROCESSOR (page 313)

Legacy option to ensure proper statistics on multiple-processor hardware.

COMPATIBILITY NO_ADREXP_CHECK (page 315)

Reverts the change in which a state variable is set so that an attempt to evaluate an address expression based on a *DODA* field will return an error instead of crashing the server.

COMPATIBILITY NO_ATODEP (page 316)

Reverts the behavior concerning indices' *TRANDEP*/*RSTRDEL* attributes during a rebuild or compact call without *XCREblk* support.

COMPATIBILITY NO_AUTO_SKIP (page 316)

Disables support for treating *TRANDEP* files as though `SKIP_MISSING_FILES` is turned on.

COMPATIBILITY NO_BLOCK_KILL (page 316)

Disables the ADMIN ability to kill currently connected clients.

COMPATIBILITY NO_CHECKFIX (page 316)

Reverts to the behavior before 2002 when the **ADDREC()** routine checks to see if a record begins with either a delete flag (0xFF) or a resource mark (0xFEFE).

COMPATIBILITY NO_CHKMBRNAMLEN (page 316)

Reverts to the original behavior to address the situation where names may have been truncated but no conflicts arose and would now get error **SNAM_ERR** at create time.

COMPATIBILITY NO_CLSTRAN_OPEN (page 317)

Reverts to the behavior of the recovery process, which was modified such that a *CLSTRAN* log entry will attempt to open the file (if it is not already opened).

COMPATIBILITY NO_COMMIT_READ_LOCK (page 317)

Disables commit read lock support for backward compatibility.

COMPATIBILITY NO_DATAMAP_CHECK (page 318)

Disables the check for mapping to the same data file as the currently-mapped file.

COMPATIBILITY NO_FLUSH_DIR (page 319)

Reverts to the old behavior which does not force the flushing of metadata to disk immediately after creates, renames, and deletes of transaction log files and transaction-dependent files.

COMPATIBILITY NO_INIT_VSPACE (page 319)

Reverts the behavior introduced in V8.14 concerning new space reserved at the end of a variable-length file and the entry that marked the preimage space as deleted if the transaction was aborted.

COMPATIBILITY NO_KEEP_OUT_TRNSEQ (page 320)

Reverts behavior that allows **TRANEND()** to distinguish between a lock obtained within a prior transaction, and a lock obtained in its own transaction.

COMPATIBILITY NO_MYMARKS (page 320)

Disables the new approach that immediately detects when all the key-level locks belong to the calling thread, and returns without checking each of the key-level locks enhancing performance. For internal FairCom testing only.

COMPATIBILITY NO_NXTMARKS (page 320)

Disables the new behavior of the cleanup routine that only performs the clean up necessary to satisfy the **NXTKEY()** operation. For internal FairCom testing only.

COMPATIBILITY NO_RELBUF_CHECK (page 321)

Disables a check for removing an updated buffer from the update list during a checkpoint was enabled.

COMPATIBILITY NO_SHUTDOWN_DELAY (page 321)

Forces an instant shutdown without pause for client disconnect.

COMPATIBILITY NO_SIGNAL_HANDLER (page 321)

Forces the server to skip the installation of signal handlers at server startup.

COMPATIBILITY NO_SMART_SAVE (page 321)

Disables support to enable a **ctsave()** call to skip the file system sync if no bytes have been written since the last call to **ctsave()**.

COMPATIBILITY NO_SPCMGT_QUEUE (page 321)

Disables a dedicated background thread performs the space reclamation on deleted member files of a Superfile and recovered variable-length data files.

COMPATIBILITY NO_SYS_FLUSH_ON_CLOSE (page 321)

Disables extra protection that causes a system cache flush before physically closing transaction logged files or write-through files.

COMPATIBILITY NO_TEST_LOCAL (page 322)

Turns off the check of whether a file is local or remote.

COMPATIBILITY NO_TRAN_DISCONNECT (page 322)

Disables support for the server disconnecting the client associated with the transaction if the need for more logs is caused by a pending transaction,

COMPATIBILITY NO_TRANDEP_SCAN (page 322)

Disables additional transaction log pre-scanning done for improved handling of transaction-dependent file renames and deletes.

COMPATIBILITY NO_UNIQFILE (page 322)

Disables attempts to determine if files accessed with different file names (or paths) and identical c-treeACE file IDs are the same file or different files.

COMPATIBILITY NO_VAR_PEOF (page 323)

Disables logic that attempts to extend the physical file size and retry growing the record in place when the growth would exceed the PEOF.

COMPATIBILITY NO_VARLEN_TRAN_UNUSED (page 323)

reverts to the previous differentiated behavior in which space reclamation for *TRANPROC* and non-*TRANPROC* files is treated differently.

COMPATIBILITY NONE (page 324)

Used in conjunction with the tamper-proof settings file to block keyword from being used in a subsequent stage of configuration loading.

COMPATIBILITY OPEN_SHARE_RW (page 325)

Restores the previous behavior of opening the files with read/write share access.

COMPATIBILITY PUTHDR_COMMIT (page 325)

Disables the PUTHDR() behavior of creating pre-image space entries, restoring old value on abort, setting the new value at the time of the call, except that on commit the header value is not reset to the new value.

COMPATIBILITY RANGE_NO_NXTKEY (page 325)

Disables the default of using **NXTKEY()** instead of **GTKEY()** to skip over records that do not meet range and/or filter criteria during range operations.

COMPATIBILITY REPLICATION_TRAN_LIST (page 325)

Makes the checkpoint's beginning-log-position-list for active transactions included in checkpoints.

COMPATIBILITY REVERT_TO_V6HDR (page 326)

Reverts to the V6 mode of only enabling extended headers for all newly created files with calls from an *Xtd8* specific function and defining the *XCREblk* structure.

COMPATIBILITY REWRITE_KEY_ERROR (page 326)

Disables the new behavior of the rewrite routines returning an error on a failed key assembly by default.

COMPATIBILITY SETEXCABT (page 326)

Reverts to the previous approach of using a loop that repeatedly attempts to update the abort node list, once for each key-level lock in the node. For internal FairCom testing only.

COMPATIBILITY SPCMGT_INDEX (page 326)

Forces c-treeACE to disable the reclamation of unused space management indices with *VLENGTH*, *TRNLOG* data files

COMPATIBILITY STREAM_FILES (page 327)

Forces c-treeACE to ignore the `LOCAL_DIRECTORY` configuration for stream files.

COMPATIBILITY SYNC_LOG (page 327)

DEPRECATED - Instructs the c-treeACE Server to open its transaction logs in synchronous write (direct I/O on Solaris) mode. See `COMPATIBILITY LOG_WRITETHRU`.

COMPATIBILITY TCPIP_CHECK_DEAD_CLIENTS (page 327)

Causes the c-treeACE Server detects when a TCP/IP client has dropped.

COMPATIBILITY TEMP_INDEX_ERROR (page 328, **http://docs.faircom.com/doc/ctserver/#49139.htm**)

Disables behavior of ignoring a **KDUP_ERR** on a temp index and setting *sysiocod* to **IDUP_COD** (-837). Also affects temporary index behavior for **ITIM_ERR** on record reads and **KDEL_ERR** on record deletes.

COMPATIBILITY USE_CHARUPPER (page 328)

Changes the default method of executing **toupper()** by optionally using **CharUpper()** on the Windows operating system.

COMPATIBILITY V24LOGON (page 328)

Legacy option that controls whether or not to use old logon connection approach.

COMPATIBILITY VDLFLG (page 328)

Re-establishes the old behavior of simply reporting the error (and abandoning a **NEWVREC()** operation) when a space-management index entry does not point to available space in a variable-length file.

COMPATIBILITY 6BTRAN_NOT_DEFAULT

```
COMPATIBILITY 6BTRAN_NOT_DEFAULT
```

Six-byte transaction number support is the default with the c-tree Server. For compatibility with existing older applications, this keyword is provided to disable this default.

See Also

```
COMPATIBILITY EXTENDED_TRAN_ONLY (page 309)
DIAGNOSTICS EXTENDED_TRAN_NO (page 349)
```

COMPATIBILITY ABORT_ON_CLOSE

```
COMPATIBILITY ABORT_ON_CLOSE
```

Disable “deferred close” capability. Force a transaction abort when a file altered in a transaction is closed before the transaction is committed.

Default: Defer close

COMPATIBILITY BATCH_SIGNAL

```
COMPATIBILITY BATCH_SIGNAL
```

This feature causes *sysiocod* to be set to **BTNO_COD** (-844) on calls to **BATSETX()**/**BATSET()** that result in the current batch being closed. If the client application finds *sysiocod* set to **BTNO_COD**, then the application does not need to issue a *BAT_CAN* call before issuing a new batch request.

COMPATIBILITY BATCH_UTRFMKEY

```
COMPATIBILITY BATCH_UTRFMKEY
```

Documentation states that keys returned with **BATSET()** called with *BAT_RET_KEY* are returned in native index order, however, **uTFRMKEY()** was called to put the keys back into the form of the client. While this behavior was changed to match the documentation, this configuration option reverts to this prior behavior.

COMPATIBILITY BLOCK_DDSFM_CREATE and BLOCK_DDSFM_DELETE

The c-treeACE Server can block adds and deletes of superfile members during the course of a dynamic dump using these two server configuration keywords.

```
COMPATIBILITY BLOCK_DDSFM_CREATE
```

Blocks superfile member creation during a dynamic dump. Attempts to create a superfile member return **DDCR_ERR** (740) with this keyword activated.

```
COMPATIBILITY BLOCK_DDSFM_DELETE
```

Blocks superfile member deletion during a dynamic dump. Attempts to remove a superfile member return **DDDR_ERR** (741) with this keyword activated.

> **Note:** An application may create or delete superfile members in a superfile host waiting to be dumped while the overall dump is going on. Once the dynamic dump begins dumping the superfile host, blocked operation cannot occur until the end of the entire dump, not just the end of the dump of the superfile host itself. Therefore, the last superfile host listed in the dump script file list will have the shortest blocking period.

Default: Do not block creates and deletes

COMPATIBILITY CHAR_SCHSEG

Schema segments automatically map a c-tree field type to a key segment type (mode). Starting with V8.14, *CT_CHAR* fields are now treated as a signed, one-byte integer, whereas, previously, they were treated as a *REGSEG* segment mode. *CT_CHARU* remains treated as an unsigned one-byte field.

To revert this change for compatibility with versions prior to V8, use the `COMPATIBILITY CHAR_SCHSEG` server configuration keyword.

```
COMPATIBILITY      CHAR_SCHSEG
```

COMPATIBILITY CHECKPOINT_OVERLAP

```
COMPATIBILITY CHECKPOINT_OVERLAP
```

A change to the checkpoint logic now causes checkpoint requests to wait for the completion of an in-process checkpoint before beginning even flushing operations. If the checkpoint is called from the **CTCHKPNT()** API, and it finds an in-process checkpoint, then it simply returns with NO_ERROR and sets *sysiocod* to **OCHK_COD** (-885). If an internal checkpoint request must

wait, a counter is incremented; and the total count is reported in *CTSTATUS.FCS* at the end of the final checkpoint (if it is nonzero). This keyword disables the new behavior.

COMPATIBILITY CHKPNT_FLUSHALL

```
COMPATIBILITY      CHKPNT_FLUSHALL
```

Forces all checkpoints but the final checkpoint to use a single system call to flush file system buffers instead of individual calls for each file to **ctsave()**. The motivation is that a call to flush the buffers for a specified file may require significant time whether or not the file has many updated pages in the file system cache; hence one flush of all the buffers may be faster than repeated calls for individual files. For Unix systems this implementation automatically invokes **sync()** unless another definition is already in place. No other platform has a default definition.

COMPATIBILITY CHKPNT_MUTEX_REL

```
COMPATIBILITY CHKPNT_MUTEX_REL
```

Forces transaction related mutexes to be released just before the checkpoint is written to disk instead of after. This should modestly reduce the time these mutexes are held by the checkpoint. It is safe because the mutexes are not released until the checkpoint has been completely composed, and its space in the log file is assigned.

Note: For FairCom internal use only.

COMPATIBILITY CLSFIL_ISAM

```
COMPATIBILITY CLSFIL_ISAM
```

By default, low-level commands are prevented from closing files opened with ISAM-level opens. This keyword is used to revert the behavior prior to V10 in which low-level commands were allowed to close files opened with ISAM-level opens.

COMPATIBILITY CLSFIL_UNBLOCK

```
COMPATIBILITY CLSFIL_UNBLOCK
```

A **FCLS_ERR** (24) could occur on a file unblock that occurred within a transaction for a file whose closure had been deferred until the transaction ends. Previous behavior was that the file had to be closed prior to the unblock call. With this configuration option it is permissible for the file blocker to have the file open at the time of the unblock call. The open may be explicit or, it may be as a result of a pending close.

COMPATIBILITY COMMPORT5000

```
COMPATIBILITY COMMPORT5000
```

Sets the default base value to add to the `SERVER_NAME` ASCII sum for determining the TCP/IP port number.

For example, the sum of the ASCII characters in FAIRCOMS equals 597. Added to the default base of 5000 results in a default TCP/IP port number of 5597.

See Also

- SERVER_NAME (page 171, http://www.faircom.com/doc/ctserver/27897.htm)
- SERVER_PORT (page 172, http://www.faircom.com/doc/ctserver/48603.htm)

COMPATIBILITY CTREE_RWLOCK

```
COMPATIBILITY CTREE_RWLOCK
```

Reader/writer lock support for Windows systems was implemented, and the mutex on the memory file hash lists was changed to a reader/writer lock when this support is enabled at compile time.

When the c-tree Server starts up on a Windows system, it checks if the system supports reader/writer lock functions. If so, it uses the system's reader/writer lock functions. If not, it uses c-tree's reader/writer lock functions. This option can be used to force c-treeACE to use custom reader/writer lock support instead of Windows' reader/writer lock support even if the Windows system supports reader/writer locks, for example, older versions not supporting this synchronization technique.

The following *CTSTATUS.FCS* messages are logged indicating the supported reader/writer locks:

```
c-tree reader/writer lock support enabled (keyword specified)
c-tree reader/writer lock support enabled (no O/S support)
Windows native reader/writer lock support enabled
```

COMPATIBILITY DIR_BUF_RQS

```
COMPATIBILITY DIR_BUF_RQS
```

During performance analysis it was noticed that get index node routine was performing retries when we expect that the node is already in a buffer. Changes were made to (a) avoid retries when the node holds the desired buffer and (b) skip the queue and directly request the holding semaphore. This option disables this new approach.

See Also

- COMPATIBILITY NO_IDXBUFBLK

Note: This is intended for internal FairCom testing only.

COMPATIBILITY DUPL_ERR_FATAL

```
COMPATIBILITY DUPL_ERR_FATAL
```

Rebuild behavior was changed such that in the event of a **DUPX_ERR**, **KDUP_ERR**, **KSRL_ERR**, or some error other than **DUPJ_ERR** or **DUPL_ERR**, one or more of the indices will have their corrupt flags turned on. As it was, **DUPJ_ERR** will not cause corrupt flag to be turned on. This option forces the index corrupt flag to be set.

COMPATIBILITY ENCRYPT128

```
COMPATIBILITY ENCRYPT128
```

c-tree's advanced encryption uses a master encryption key to encrypt the following items using the AES cipher:

1. The security resource in c-tree data and index files that use advanced encryption, and
2. The encryption key in the transaction log file header when using advanced log encryption.

The length of the master key was increased from 128 bits to 256 bits. c-treeACE can still access files that created with the previous 128-bit master key. This keyword reverts to the original 128-bit master key usage for encryption.

Attempting to open a file created using a 256-bit master key fails with error **ENCK_ERR** (952) when 256-bit key support is disabled.

See Also

- COMPATIBILITY LOG_ENCRYPT128 (page 312)

COMPATIBILITY ESTIMATE_SCALE

```
COMPATIBILITY ESTIMATE_SCALE
```

SQL index selection relies heavily on the precision of **ESTKEY()** to pick the best index. Prior behavior was for **ESTKEY()** to return the estimate to within 1/10 percent. New behavior increases the **ESTKEY()** precision with the number of keys in the file to attempt to maintain a precision of about 100 records. As this is done using a binary search on the high and low keys, it requires 2 additional index searches for each doubling of the file size. This option reverts to the prior behavior.

COMPATIBILITY EXACT_FILE_NAMES

```
COMPATIBILITY EXACT_FILE_NAMES
```

Force different references to the same file to use the same names. For example: *C:\data\temp.dat* and *\data\temp.dat* would be considered different even if they referred to the same file.

Default: Allow different names

COMPATIBILITY EXTENDED_TRAN_ONLY

```
COMPATIBILITY EXTENDED_TRAN_ONLY
```

This keyword forces a **R6BT_ERR** (745) on an attempt to create or open a non-extended-transaction-number file. A read-only open is not a problem since the file cannot be updated.

See Also

COMPATIBILITY 6BTRAN_NOT_DEFAULT (page 305)
DIAGNOSTICS EXTENDED_TRAN_NO (page 349)

COMPATIBILITY FAILED_TRAN_IO

COMPATIBILITY FAILED_TRAN_IO

When transaction processing is in effect, a failed write operation for transaction controlled files terminates the server (or stand-alone application) with error **L60** (*ctcatend*). This error can occur with any write operation errors of buffer and cache pages, and header write operations for transaction controlled files. If an **L60** error does occur, the *Mx* value will be important in determining the precise location of the error.

This keyword restores behavior prior to this change.

COMPATIBILITY FILE_CREATE_CHECKPOINT

COMPATIBILITY FILE_CREATE_CHECKPOINT

The old behavior was to generate a checkpoint if a TRNLOG (but not TRANDEP) file is about to be deleted or renamed after a file, any file, is created but with no intervening checkpoint. That is, the sequence Create FileA, Checkpoint, Create FileB, Delete FileA would cause a second checkpoint during the delete processing of FileA even though the FileA create preceding the delete occurred with an intervening checkpoint. With new default behavior a checkpoint is generated only if the file to be renamed or deleted was the one created without an intervening checkpoint. This keyword option disables this behavior.

COMPATIBILITY FILE_DESCRIPTOR_LIMIT

COMPATIBILITY FILE_DESCRIPTOR_LIMIT

Running c-treeACE Server with insufficient file descriptors can lead to errors opening files. See FILES (page 167).

This compatibility keyword overrides the behavior described in FILES (page 167) in case it is not convenient for a system administrator to set the file descriptor limit for the c-treeACE Server process to the required value or it is not desired to decrease the FILES or CONNECTIONS settings.

> **Note**: Using this keyword is not recommended. It is provided for backward compatibility or short-term use until the administrator is able to increase the file descriptor limit for the c-treeACE Server process.

A message is logged to *CTSTATUS.FCS* explaining that the `COMPATIBILITY FILE_DESCRIPTOR_LIMIT` configuration option can be used to allow the server to start in this situation:

```
Tue Apr 29 12:23:44 2014
 - User# 00001  ERROR: The hard limit on file descriptors available to this process (500) is lower
```

```
than the database engine's file descriptor requirement (1043). Either increase the hard limit,
or decrease the FILES or CONNECTIONS settings.
Tue Apr 29 12:23:44 2014
 - User# 00001  Note: The configuration option COMPATIBILITY FILE_DESCRIPTOR_LIMIT can be used
to allow c-tree Server to start even if the file descriptor limit is insufficient. However, this
can lead to errors opening files.
```

See Also

- *FILES* (page 167)

COMPATIBILITY FILELIST_GROWTH

```
COMPATIBILITY FILELIST_GROWTH
```

When a key search occurs on partitioned indexes that are not ordered as the partition key, all of the partition members must be opened, which can take a significant amount of time if the number of partitions is large. One operation where unnecessary time was spent was resizing (allocate new + copy + free old) the users file control block. Previously, this grew in increments of 32 (MAXMEMB + 1) files. With partition files used through SQL, thousands of files may be opened by a single query. A change was made which increased the rate at which this array grows by doubling in size up to 2 thousand files. This change reduced the total query time by 4% in a case with 1000 files. This will affect memory usage, as it will now increase in increments of up to 500 KB (for large numbers of files already opened) versus 8KB with the old approach. This option reverts to the old approach.

COMPATIBILITY LFL_WAIT

```
COMPATIBILITY LFL_WAIT
```

Eliminates the lflsema SHORTWAIT loop, and uses a call with the WAIT parameter.

Note: For internal FairCom use only.

COMPATIBILITY LFW_ADAPTIVE

```
COMPATIBILITY LFW_ADAPTIVE
```

Replaces the commit delay **ct_udefer()** call with a loop that permits the "prime" cohort to periodically wake up, after a much smaller defer time, and check for commit delay progress. `COMMIT_DELAY_USEC` determines the length of the "short" sleep when this option is enabled.

Note: For internal FairCom use only.

COMPATIBILITY LOCK_CACHE

```
COMPATIBILITY LOCK_CACHE
```

Allows c-tree attempts to lock all data and index cache pages into physical memory, using the **mlock()** function on Unix systems. In the event this fails, the server will fail to initialize and a message is printed to *CTSTATUS.FCS:*

```
LOCK_CACHE: mlock() error 12
```

This is only implemented for Unix, and a different approach will be required to implement on Windows, as this behavior is accomplished through the Microsoft **VirtualAlloc()** function.

Note: This feature is currently disabled.

COMPATIBILITY LOCK_HEADER

```
COMPATIBILITY LOCK_HEADER
```

Some header calls always acquire the header semaphore. It is possible the associated lock table entries are not needed since the semaphore provides access control. With new default behavior, lock calls skip the lock table entries, only acquiring the header semaphore. This option will restore the lock table entries.

COMPATIBILITY LOG_ENCRYPT128

```
COMPATIBILITY LOG_ENCRYPT128
```

c-tree's advanced encryption uses a master encryption key to encrypt the following items using the AES cipher:

1. The security resource in c-tree data and index files that use advanced encryption, and
2. The encryption key in the transaction log file header when using advanced log encryption.

The length of the master key was increased from 128 bits to 256 bits. c-treeACE can still access files that created with the previous 128 bit master key. This keyword reverts to the original 128-bit master key usage.

Attempting to read a transaction log encrypted with a 256 bit encryption key when only 128-bit log encryption support is enabled fails and c-tree logs the following message to *CTSTATUS.FCS*:

```
Cannot read existing encrypted logs.  Log is using 256 bit encryption key but server is configured
for 128 bit encryption support.
```

See Also

- COMPATIBILITY ENCRYPT128 (page 309)

COMPATIBILITY PUTHDR_COMMIT

```
COMPATIBILITY PUTHDR_COMMIT
```

A **KDUP_ERR** (2, duplicate key found) could be encountered while using a *SRLSEG* value as the key. The *SRLSEG* values (*sernum1*, *sernum2*) are stored in the c-tree file header. Header values are handled differently than ordinary record contents, as they are not maintained under typical transaction control; we do not want to hold a lock for the duration of a transaction for these types of values.

PUTHDR() behavior was changed such that on commit, the header value is not reset to the new value. Abort behavior remains the same. This also affects *numrec1*, *numrec2*, *phyrec1*, and *phyrec2* header values.

For backward compatibility, the original behavior can be restored by adding this keyword to the configuration file.

COMPATIBILITY MEMORY_FILE_SKIP_FREE

```
COMPATIBILITY MEMORY_FILE_SKIP_FREE
```

When a memory file is actually closed (which is not necessarily when the last user closes the file because of the *ctKEEPOPEN* attribute), the individual records or nodes are returned to the memory heap.

This keyword skips the free of memory at server shutdown for the individual memory records for memory files still open and assume that the server termination will (automatically) return the memory to the system. This may quicken the shutdown in the case of a large number of memory file records and/or nodes.

COMPATIBILITY MEMORY_LIMITS

```
COMPATIBILITY MEMORY_LIMITS
```

Legacy V7 backward compatibility option that enforces system or user memory limits.

COMPATIBILITY MULTI_PROCESSOR

```
COMPATIBILITY   MULTI_PROCESSOR
```

Legacy option to ensure proper statistics on multiple-processor hardware.

COMPATIBILITY MULTIOPN_*

Se the default for enhanced locking control for files opened multiple times in the same connection. The system-level default can be controlled by using one of the following configuration keywords which sets the behavior accordingly to their names.

```
COMPATIBILITY MULTIOPN_DIFUSR
COMPATIBILITY MULTIOPN_SAMUSR_M
COMPATIBILITY MULTIOPN_SAMUSR_1
```

c-treeACE supports opening the same file multiple times in the same connection assigning a different file number to each file or, in c-treeDB, a different file handle. Each of these sibling files is referred to as a "co-file." For example, if the file *customer.dat* is opened in the same connection using file numbers 5 and 10, then we say that file 5 is a co-file of file 10, and vice versa.

In this case there are considerations about how locks interact within the same connection when operating using different co-files. For example, if a write lock is acquired on a record R using file number 5 within the same connection, what is the behavior of trying to acquire a lock on R using co-file number 10?

In this example, before this enhancement, c-treeACE Server behaved as follows:

The lock on R issued with co-file number 10 succeed and is considered a “secondary lock”, while the lock acquired first (using file number 5) is considered "primary."

The difference in the locks manifests itself during calls to unlock the record: If the primary lock is unlocked first, then the primary lock and all the corresponding locks on co-files are removed. But if a secondary lock is unlocked before the primary lock is unlocked, then only the secondary user lock is removed; and the primary lock is maintained.

Any other connection saw the record locked until the primary lock gets released.

This previous behavior has been maintained and it is the system-level default behavior.

It is now possible to configure the behavior choosing among 4 different options:

- **NODIFUSR**: The default as described above.
- **DIFUSR**: Locks on co-files are considered as acquired from a different connection, so the lock on R issued with co-file number 10 will fail.
- **SAMUSR_M**: Locks on record R on co-files are considered as the same lock acquired on the same file, so lock on R issued with co-file number 10 succeeds. As soon as the lock is released in one of the co-files that successfully requested the lock, the lock is released. Therefore before acquiring the lock on R using file number 10, the lock can be released only using file number 5, but after acquiring the lock on R using file number 10, the lock can be released either by using file number 5 or 10.
- **SAMUSR_1**: Locks on record R on co-files are considered as the same lock acquired on the same file, so lock on R issued with co-file number 10 succeeds. As soon as the lock is released in one of the co-files (whether or not the lock was requested using the co-file) the lock is released. Therefore even before acquiring the lock on R using file number 10 the lock can be released either by using file number 5 or 10.

Recursive locks are not supported for co-files. An attempt to open a co-file when recursive locks are pending on the underlying file will fail with the error **MUOP_RCR** (998). An attempt to issue a lock on a co-file with the *ctLK_RECR* bit set in the lock mode will fail with the error **MLOK_ERR** (999).

Read locks behave in a manner consistent with write locks. The notable issues are:

1. With DIFUSR, read locks can be issued for different co-files; and unlocking one co-file's read lock does not remove the read lock from any other co-files that requested the read lock.
2. With DIFUSR, a read lock on a co-file cannot be promoted to a write lock if other co-files have read locks; a non-blocking write lock will fail with **DLOK_ERR** (42) and a blocking write lock will fail with **DEAD_ERR** (86).
3. With SAMUSR_*, read locks can be issued for different co-files, and unlocking one co-file read lock unlocks all the co-file read locks.
4. With SAMUSR_*, read locks can be promoted to write locks as long as no other threads have also acquired read locks.
5. With SAMUSR_1, a read lock on a co-file can be unlocked using another co-file's file number even if no lock has been issued using the other co-file number.

The system-level default can be controlled by using one of the following configuration keywords which sets the behavior accordingly to their names.

- `COMPATIBILITY MULTIOPN_DIFUSR`
- `COMPATIBILITY MULTIOPN_SAMUSR_M`
- `COMPATIBILITY MULTIOPN_SAMUSR_1`

A connection can override the system-level default for all open instances of a file by calling:

PUTHDR(*datno*, *mode*, *ctMULTIOPNhdr***)**

Where *mode* is one of the following:

- ctMULTIOPNnodifusr
- ctMULTIOPNdifusr
- ctMULTIOPNsamusr_M
- ctMULTIOPNsamusr_1

If no **PUTHDR** call is made, the system-level default is used for that connection's instances of the file. When a file is opened, if that connection already has the file open, the newly opened file inherits the MULTIOPN setting of the already-open file instance. An attempt to change the setting so that one instance of the file would be inconsistent with the others will fail with error **MOFL_ERR**. A file's MULTIOPN state can only be changed if it is the first open instance of the file and it has no pending locks.

COMPATIBILITY NLM_DEFER_THREADSWITCH

`COMPATIBILITY NLM_DEFER_THREADSWITCH`

This option can improve the performance of the c-treeACE Server for Novell at the cost of decreased performance in other processes. Consult with your application developer and Novell system administrator to determine if this switch is appropriate for your system

Default: Defer

COMPATIBILITY NLM_LONG_FILE_NAMES

`COMPATIBILITY NLM_LONG_FILE_NAMES`

Instructs the c-treeACE Server to use OS/2 namespace support, provided OS/2 namespace support is enabled on the working volume. If the keyword is not used, or if the volume does not support OS/2 namespace, long file names are not supported. FairCom recommends that when using long file name support all volumes provide OS/2 namespace support to prevent an error. This keyword is only required by the NLM c-treeACE Server and is ignored in all other versions.

Default: Not supported

COMPATIBILITY NO_ADREXP_CHECK

`COMPATIBILITY NO_ADREXP_CHECK`

r-tree virtual fields should be in "correct" order where correct means no forward references. Using a forward reference results in various errors, resulting in potential server crashes. Address expressions involving *DODA* fields can be distinguished from address expressions involving other virtual fields, and a state variable is now set in the step of the script where the address expression is executed such that during the parsing phase, an attempt to evaluate an address expression based on a *DODA* field will return an error instead of crashing the server. This keyword reverts this change for testing purposes only.

COMPATIBILITY NO_ATODEP

`COMPATIBILITY NO_ATODEP`

When indices have to be recreated during a rebuild or compact call without *XCREblk* support, the indices will lose their *TRANDEP*/*RSTRDEL* attributes. The rationale to change this behavior is to permit such calls to automatically assign the indices the same *TRANDEP*/*RSTRDEL* attributes as the associated data file. A new client, even with the new default behavior turned on will not exhibit this new behavior if **RBLIFIX8()**/**CMPIFILX8()** are called since the *XCREblk*'s passed to rebuild or compact will prevail. This option reverts this behavior.

COMPATIBILITY NO_AUTO_SKIP

`COMPATIBILITY NO_AUTO_SKIP`

To avoid requiring `SKIP_MISSING_FILES` when *TRANDEP* files are in use, a new default behavior effectively treats *TRANDEP* files as though `SKIP_MISSING_FILES` is turned on, however, for files without *TRANDEP* activities, recover, rollback, or roll-forward may still terminate execution if unexpected missing files are encountered.

This keyword disables this support to revert to the original behavior.

> **Note:** It is possible that an unexpected **FNOP_ERR** error can still occur for a *TRANDEP* file, however, this change should greatly reduce the number of unexpected **FNOP_ERR**'s.

See Also

- COMPATIBILITY NO_CLSTRAN_OPEN (page 317)

COMPATIBILITY NO_BLOCK_KILL

`COMPATIBILITY NO_BLOCK_KILL`

Disable the ADMIN ability to kill currently connected clients.

Default: Allow kill

COMPATIBILITY NO_CHECKFIX

`COMPATIBILITY NO_CHECKFIX`

The **ADDREC()** routine checks to see if a record begins with either a delete flag (0xFF) or a resource mark (0xFEFE). If so, the **ADDREC()** returns **FBEG_ERR** (553). Behavior prior to 2002 did not make this check and this option reverts the newer behavior.

COMPATIBILITY NO_CHKMBRNAMLEN

`COMPATIBILITY NO_CHKMBRNAMLEN`

The length of a c-tree Superfile name is restricted because the Superfile directory index has a restricted key length. A default behavior checks when the Superfile member name is restricted due to small page sizes: if the Superfile name is truncated in the Superfile member directory index, then the create of the member will fail with **SNAM_ERR** (418). Without this new behavior,

member names that would be truncated to the same string would cause **DOPN_ERR** (19) and/or **KOPN_ERR** (18) errors.

To address the situation where names may have been truncated but no conflicts arose and would now get error **SNAM_ERR** at create time, `COMPATIBILITY NO_CHKMBRNAMLEN` is available to revert to the original behavior.

COMPATIBILITY NO_CLSTRAN_OPEN

`COMPATIBILITY NO_CLSTRAN_OPEN`

c-treeACE is designed to permit automatic recovery and rollbacks to properly handle deleted and renamed transaction dependent files (*TRANDEP*) without requiring the `SKIP_MISSING_FILES` configuration to be enabled. Situations can occur where a *TRANDEP* file has been physically deleted (by a system call, for example) that will interfere with a transaction rollback during automatic recovery.

If `SKIP_MISSING_FILES` is active, then the rollback should succeed even with the missing file, however, should only ignore missing files for which there is a transaction controlled explanation such as a file delete or file rename.

The recovery process was modified such that a *CLSTRAN* log entry will attempt to open the file (if it is not already opened). Upon detecting a *CLSTRAN* entry triggers putting a missing file on the list of missing files. To revert to the original behavior, the server configuration keyword, `COMPATIBILITY NO_CLSTRAN_OPEN` is provided.

See Also

- COMPATIBILITY NO_AUTO_SKIP (page 316)

COMPATIBILITY NO_COMMIT_READ_LOCK

`COMPATIBILITY NO_COMMIT_READ_LOCK`

Disables commit read lock support for backward compatibility. c-treeACE V9 introduced commit read lock support as a default. Without explicit read or write locks, it is possible to have a partially updated record buffer returned in a high transaction volume environment. The returned record buffer could consist of partial old data, and partial newly updated data from a transaction commit operation from a concurrent thread. While the occurrence of this event is extraordinarily rare (on the order of one in a million record reads) it is clearly important in a high volume situation. To prevent these “dirty” record reads. Commit Read Locks enable an implicit, high performance, low-level record lock ensuring consistent data record reads in high volume transaction environments.

When explicit read or write locks are enforced in the c-tree application then this feature is not required, as the transaction commits will ensure consistent data reads.

A new type of lock entry, *CMT*, was added to the c-treeACE Server lock table. These *CMT* entries are guaranteed to be very shortly held (only for the duration of the internal read operations). Under high volume server operations this new lock type may occasionally be observed as "*forcei cmtlok*" in a **ctLOKDMP()** (**LockDump()**) output.

Commit Read Lock Errors

An attempt to update a record without an explicit lock with the commit read logic is active results in error **CMLK_ERR** (768). This lock error fails the transaction commit. **CULK_ERR** (769) indicates an unexpected failure during the removal of a commit lock. This is an extremely rare internal c-treeACE Server error and should not be encountered in standard usage.

Performance Considerations

The commit read lock uses polling logic to permit a thread to retry a commit lock when it cannot be acquired. For example, if the updater has already acquired a write commit lock, a reader will be denied its request for a read commit lock, and the reader will retry. Particularly, for large transactions, the retries can consume a large amount of CPU time.

To avoid this high CPU utilization a more adaptive retry logic was implemented. If several retries fail in succession, the retry call is changed before each retry from a defer time of 0, to a defer time of *ctredcmtdfr* where *ctredcmtdfr* defaults to 10 milliseconds.

To provide an additional tuning mechanism for this retry value, the following configuration keyword `COMMIT_LOCK_DEFER_MS  <defer time in milliseconds>`

The length of the defer value can be varied from zero to 100 milliseconds.

Internal tests demonstrated the affect of this change on CPU utilization was dramatic as a reader attempted to retry its read commit lock. Of course, actual performance increases will be variable, depending on any particular server environment. The trade-off with this method is introducing an unnecessary defer (i.e. if the next retry without a non-zero defer would have succeeded). In practice, this was not found to impede performance.

Commit write locks held by the transaction (i.e., locks that block read attempts during the actual commit process) are held during the entire commit. This has no direct impact upon the transaction commit, however, can cause longer delays for a read attempt when the transaction itself is comprised of a large number of write operations (e.g., committing thousands of **ADDREC()**s)

See Also

`COMMIT_LOCK_DEFER_MS (page 214)`

COMPATIBILITY NO_DATAMAP_CHECK

`COMPATIBILITY NO_DATAMAP_CHECK`

When an index file is opened by an ISAM file open function, if the index is already open and is associated with a data file, c-treeACE checks if the index is associated with a different data file than the data file that is involved in the current file open operation. If so, the open now fails with error **DMAP_ERR** (957, this index file is already mapped to a different data file). This keyword disables the new approach (no check for mapping to the same data file as the currently-mapped file).

COMPATIBILITY NO_DELNOD_QUEUE

`COMPATIBILITY NO_DELNOD_QUEUE`

c-treeACE Server's delete-node thread prunes empty nodes from c-tree index files in the background. This maintains index key data densely packed for optimal performance. This activity

requires directly opening the index file by the delete-node thread, which happens during idle times when it may be expected external applications can access the file. Having the file open at the time by the delete-node thread prevents external file open.

In exceptional cases, this behavior may not be desirable as external processes expect complete access to the file when it is no longer in use by the application. For example, processes may wish to immediately copy the file for other external processing.

In V11 and later, c-treeACE Server supports a configuration option to disable the internal delete-node thread. The option `COMPATIBILITY NO_DELNOD_QUEUE` disables the delete-node thread and disables the writing of entries to the delete-node queue.

Caution: This option should only be used in special situations in which you absolutely require an external process access to closed files, which is discouraged if at all possible while the server is operational.

Operating without pruning empty nodes from indexes can potentially diminish performance of certain index search operations which may non-optimally traverse many empty leaf nodes. This is especially true for applications which heavily delete records from the database.

COMPATIBILITY NO_FLUSH_DIR

```
COMPATIBILITY NO_FLUSH_DIR
```

Reverts to the old behavior which does not force the flushing of metadata to disk immediately after creates, renames, and deletes of transaction log files and transaction-dependent files.

When a file is created, renamed, or deleted, the new name of the file is reflected in the file system entry on disk only when the containing directory's metadata is flushed to disk. If the system crashes before the metadata is flushed to disk, the data for the file might exist on disk, but there is no guarantee that the file system contains an entry for the newly created, renamed, or deleted file. In a test case we noticed that after a system power loss a transaction log containing valid log entries still had the name of the transaction log template file.

In V11 and later, c-tree now ensures that creates, renames, and deletes of transaction log files and transaction-dependent files are followed by flushing of the containing directory's metadata to disk. This change also applies to other important files such as *CTSTATUS.FCS*, the master key password verification file, and files created during file compact operations (even if not transaction dependent).

To revert to the old behavior, add `COMPATIBILITY NO_FLUSH_DIR` to *ctsrvr.cfg*.

COMPATIBILITY NO_INIT_VSPACE

```
COMPATIBILITY      NO_INIT_VSPACE
```

Under transaction control, new space was reserved at the end of a variable-length file, and the preimage space received an entry that marked the space as deleted if the transaction was aborted. When a write to this space (say as part of an **ADDVREC()**) was made, still under the same transaction control, preimage received another entry. However, until either a commit or an abort, the new region appeared to other users to be simply ff-filled. When another user was scanning the variable-length file in physical order, e.g., with **NXTVREC()**, this would cause a

VFLG_ERR (158), since 0xffff is not a valid record mark. A fixed-length file does not behave this way because ff-fill appears as a deleted record to be skipped during physical order scan.

In V8.14 this behavior was changed such that not only is preimage space updated with the deleted record mark, but also a write is issued to the actual file with the same contents. This option reverts that behavior.

See also

COMPATIBILITY NO_KEEP_OUT_TRNSEQ

```
COMPATIBILITY NO_KEEP_OUT_TRNSEQ
```

ctKEEP_OUT permits **TRANEND()** to free locks obtained inside the transaction, but keep locks obtained outside of the transaction. One subtle point is that a lock obtained outside of the transaction on a record that is updated within the transaction will be freed. *ctKEEP_OUT_ALL* keeps locks even if the record is updated within the transaction if the lock was obtained outside of the transaction. However, a lock obtained inside of a prior transaction that is kept at the **TRANEND()** of the prior transaction is treated as if it was obtained inside of a subsequent transaction , and is released at the next **TRANEND()** called with *ctKEEP_OUT* or *ctKEEP_OUT_ALL*.

New default behavior allows **TRANEND()** to distinguish between a lock obtained within a prior transaction, and a lock obtained in its own transaction. This option reverts this behavior.

COMPATIBILITY NO_MYMARKS

```
COMPATIBILITY NO_MYMARKS
```

In some situations repeated calls were made to to resolve key-level locks without any clean up occurring. When a key-level lock is associated with a transaction that has committed or aborted, the key-level lock is removed and the key value stays (committing an add or aborting a delete) or is removed (committing a delete or aborting an add). But if a thread is executing a long transaction, it may be repeatedly calling for cleanup without any effect when the key-level locks for a node all belong to the calling thread.

Buffer state variables were added such that this cleanup can immediately detect when all the key-level locks belong to the calling thread, and returns without checking each of the key-level locks enhancing performance. This keyword disables the new approach for testing.

Note: This is intended for internal FairCom testing only.

COMPATIBILITY NO_NXTMARKS

```
COMPATIBILITY NO_NXTMARKS
```

To improve the efficiency of key cleanup when called as part of a **NXTKEY()** operation, the cleanup routine was modified to only perform the clean up necessary to satisfy the **NXTKEY()** operation. If **NXTKEY()** ends up calling **GTKEY()** because the last position for the calling thread is no longer valid, then the subsequent clean up would not use the special next key logic. This keyword disables this new behavior.

Note: This is intended for internal FairCom testing only.

COMPATIBILITY NO_RELBUF_CHECK

```
COMPATIBILITY NO_RELBUF_CHECK
```

A check for removing an updated buffer from the update list during a checkpoint was enabled. This new behavior can be disabled with this keyword.

COMPATIBILITY NO_SHUTDOWN_DELAY

```
COMPATIBILITY NO_SHUTDOWN_DELAY
```

Forces an instant shutdown without pause for client disconnect. Not valid on NLM.

Default: Wait for client

COMPATIBILITY NO_SIGNAL_HANDLER

```
COMPATIBILITY NO_SIGNAL_HANDLER
```

Developers using the Server SDK to build custom servers with their own over all control of the server can use this server configuration keyword to force the server to skip the installation of signal handlers at server startup. If this is done, then their control code will be responsible for dealing with signals. This modification only affects Unix platforms. If signal handlers are implemented for other platforms, then be sure that the compatibility test in *ctsint_a.c* is repeated within the appropriate platform #ifdef.

COMPATIBILITY NO_SMART_SAVE

```
COMPATIBILITY  NO_SMART_SAVE
```

Disables support to enable a **ctsave()** call to skip the file system sync if no bytes have been written since the last call to **ctsave()**.

COMPATIBILITY NO_SPCMGT_QUEUE

```
COMPATIBILITY NO_SPCMGT_QUEUE
```

By default, the c-treeACE Server reclaims the space from deleted member files of a Superfile and recovered variable-length data files. A dedicated background thread performs the space reclamation. A permanent queue stored in the Server file *D0000001.FCS* permits the space reclamation to be interrupted at Server shutdown, and resumed when the Server restarts. This configuration keyword disables this feature.

Default: Manage Superfile deleted space

COMPATIBILITY NO_SYS_FLUSH_ON_CLOSE

```
COMPATIBILITY NO_SYS_FLUSH_ON_CLOSE
```

In some situations, it is possible for a file-close operation to occur without a guarantee of a data sync to disk first. For transaction-controlled files, this meant a file could be left corrupted on disk while the c-treeACE Server transaction control system would have no knowledge of this. Extra protection has been added to ensure that this situation did not occur.

This behavior will cause a system cache flush before physically closing transaction logged files or write-through files. This behavior can be disabled by adding `COMPATIBILITY NO_SYS_FLUSH_ON_CLOSE` to the server configuration file.

COMPATIBILITY NO_TEST_LOCAL

`COMPATIBILITY NO_TEST_LOCAL`

In some environments (e.g., WIN32 / UNIX) the tests to determine if two files with different names are really different (or just accessed with different paths or aliases, etc.) may indicate the files are different, when in fact they are the same, if one of the names is based on a network reference and the other (through aliases, device mappings or SUBST commands) appears like a local file. If this occurs, the server may attempt to open the files as two physically different files.

Because of the possibility of a performance hit, `COMPATIBILITY NO_TEST_LOCAL` is available to turn off the check of whether a file is local or remote.

COMPATIBILITY NO_TRAN_DISCONNECT

`COMPATIBILITY NO_TRAN_DISCONNECT`

When the number of log files is not permitted to increase (because of `FIXED_LOG_SPACE YES` in the configuration), and if the need for more logs is caused by a pending transaction, the server disconnects the client associated with the transaction by default. This keyword option disables this support and if the client does not make a subsequent server request, then the pending transaction will eventually lead to the server terminating abruptly with a L56 ctcatend. The server terminates because it cannot ensure that a commit or abort will be added to the transaction logs before the log that holds the **TRANBEG()** will become inactive. (If the client makes a server request, it will see the transaction attribute that indicates the need to abandoned the transaction, and the ctcatend shutdown will be avoided.)

COMPATIBILITY NO_TRANDEP_SCAN

`COMPATIBILITY NO_TRANDEP_SCAN`

Additional transaction log pre-scanning is now done for improved handling of transaction-dependent file renames and deletes, avoiding potential **LEOF_ERR** or **ITIM_ERR** errors. This option disables this pre-scan feature.

COMPATIBILITY NO_UNIQFILE

`COMPATIBILITY NO_UNIQFILE`

Disables attempts to determine if files accessed with different file names (or paths) and identical c-treeACE file IDs are the same file or different files.

Default: Check file identity

COMPATIBILITY NO_VAR_PEOF

```
COMPATIBILITY  NO_VAR_PEOF
```

A default behavior permits a variable-length record that grows to grow in place if (a) the record is at the Logical End of File (LEOF), (b) no other user is adding a record at the LEOF, and (c) the growth of the record does not exceed the current Physical End of File (PEOF). Additional behavior (default) removes the limitation that the record growth stay within the current PEOF.

When the growth would exceed the PEOF, the logic attempts to extend the physical file size and retry growing the record in place (within the newly extended PEOF). At runtime, `COMPATIBILITY NO_VAR_PEOF`, disables this support, and leaves the growth in place activated as long as the record fits within the current PEOF.

COMPATIBILITY NO_VARLEN_TRAN_UNUSED

```
COMPATIBILITY  NO_VARLEN_TRAN_UNUSED
```

Previously, there were two related aspects of variable record length space reclamation that differed for transaction-controlled files. Instances were noted where a *TRANPROC* file would grow unnecessarily. In a particular instance, frequent and constant c-treeACE SQL add and delete operations resulted in substantial growth of a data file.

To avoid this unexpected growth, both aspects of space reclamation are now treated the same for *TRANPROC* and non-*TRANPROC* files. `COMPATIBILITY NO_VARLEN_TRAN_UNUSED` reverts to the previous differentiated behavior.

COMPATIBILITY NO_VFLG_ERR

Disables new handling of **NXTVREC** in physical order when it encounters the "record space" for a new record that has not been committed and was written in a space that is not being reused. Restores the earlier (post V8.14) behavior for the rare circumstance in which the old behavior is desired.

A feature introduced in V8.14 affected the behavior of **NXTVREC** in physical order when it encountered the "record space" for a new record that had not been committed and was written in a space that was not being reused. Instead of reporting a **VFLG_ERR** (error 158), that new feature skips the uncommitted "record space" (unless the reading is by the transactor, which would see the uncommitted record).

Details

The change affects how variable-length records are internally marked during transaction processing of pre-image space. The behavior prior to V8.14 marked the record header in a way that was considered invalid, causing a **VFLG_ERR** (158) error. The newer behavior sees the record as a deleted record (actually a pending insert) and skips to the next record as in a "read committed" transaction isolation.

This change is limited to the internal handling of header marks for newly added variable-length records.

Indexed files are not affected by these changes because pending key inserts are handled differently under transaction control.

These changes do not include changes to the physical files, record structures on disk, or other transaction control.

Reverting Back to the Old Behavior

These changes can be reverted back to the original (prior to V8.14) behavior using the keyword:

`COMPATIBILITY NO_INIT_VSPACE`
(`http://www.faircom.com/doc/ctserver/#57570.htm`)

Changes in the Latest Revision

The changes introduced in the V11 release address the issues with the earlier change as follows:

If the reader has requested acquiring locks on the records that it reads, the physical read acquires a lock on that record and respects the lock before proceeding (earlier the lock was not respected) producing one of the following outcomes:

If...	Then...
Locking the record fails with error 42 (**DLOK_ERR**) because another connection has the record locked and the reader requested non-blocking locking.	The physical record read function returns error 42.
The record is committed (already, or when the lock is released).	The record read proceeds as usual, and the record that would have been skipped is returned to the caller.
The record is deleted or is a “resource.”	The physical record read function continues scanning the data file, reading the next record.
The record header contains an invalid record mark.	The physical record read function returns error 158 (**VFLG_ERR**).

Further changes have been introduced to reduce the occurrence of the 158 (**VFLG_ERR**) without skipping any record by changing the record header marker management during record addition.

`COMPATIBILITY NO_VFLG_ERR` can be used to disable this new handling and restore the earlier (post V8.14) behavior for the rare circumstance in which the old behavior is desired.

COMPATIBILITY NONE

`COMPATIBILITY NONE`

This option is used in conjunction with the tamper-proof settings file under the server. Configuration options that are in the encrypted *ctsrvr.set* settings file cannot be overridden in the *ctsrvr.cfg* file.

The DIAGNOSTICS, COMPATIBILITY, and CONSOLE keywords do not automatically block use in a subsequent stage of configuration loading. To explicitly block any of these keywords present in a later stage, add entries in the form: `<keyword> NONE` where `<keyword>` is `DIAGNOSTICS`, `COMPATIBILITY`, or `CONSOLE`. For example, to turn on the abort-on-close compatibility option and prevent any subsequent stage to use the `COMPATIBILITY` keyword, place the following entries in *ctsrvr.set*:

```
COMPATIBILITY ABORT_ON_CLOSE
```

```
COMPATIBILITY NONE
```

Default: Not present

COMPATIBILITY OPEN_SHARE_RW

```
COMPATIBILITY OPEN_SHARE_RW
```

For c-tree data and index files that do not use the *DUPCHANEL* filemode, the c-treeACE for Windows now opens the files in shared read-only mode such that only the c-treeACE process can write to the files. **FNAC_ERR** (920), is returned when a file exists but is not accessible, say due to file system permission settings or file sharing restrictions. This configuration option can be used to restore the previous behavior of opening the files with read/write share access.

COMPATIBILITY OPEN_RANDOM_ACCESS

```
COMPATIBILITY OPEN_RANDOM_ACCESS
```

Restores the behavior of the obsolete `FILE_FLAG_RANDOM_ACCESS` option.

COMPATIBILITY PUTHDR_COMMIT

```
COMPATIBILITY PUTHDR_COMMIT
```

PUTHDR() called within a transaction causes a pre-image space entry that contains the old and new header member value. On an abort, the old value is restored. On commit the new value is reset. "Reset" because at the time of the call to **PUTHDR()**, the header member takes on the new value. But header values are not like ordinary record contents. They are not typically under transaction control. We do not want to lock the record header during a transaction and not unlock until the commit. **PUTHDR()** uses the more traditional transaction approach to permit special modifications to the file under transaction control. However, for the *ctSERNUMhdr*, *ctLOGEOFhdr* and *ctPHYEOFhdr* header members, this approach can lead to unexpected problems as these entries are always increasing and have an aspect of uniqueness about them. For *ctSERNUMhdr*, *ctLOGEOFhdr* and *ctPHYEOFhdr*, **PUTHDR()** behaves as before (creating pre-image space entries, restoring old value on abort, setting the new value at the time of the call) except that on commit the header value is not reset to the new value. The prior behavior can be restored by adding this keyword option.

COMPATIBILITY RANGE_NO_NXTKEY

```
COMPATIBILITY RANGE_NO_NXTKEY
```

Range performance can be enhanced using **NXTKEY()** instead of **GTKEY()** to skip over records that do not meet range and/or filter criteria during range operations. A modification was made to enable this behavior by default. This option disables this feature.

COMPATIBILITY REPLICATION_TRAN_LIST

```
COMPATIBILITY REPLICATION_TRAN_LIST
```

Makes the checkpoint's beginning-log-position-list for active transactions included in checkpoints. This also applies to the pending SUCTRAN list.

COMPATIBILITY REVERT_TO_V6HDR

```
COMPATIBILITY REVERT_TO_V6HDR
```

Version 9 of c-treeACE introduced extended headers for all newly created files by default. Previously, this mode was only enabled with calls from an *Xtd8* specific function and defining the *XCREblk* structure.

The advantage of this new approach is that 6-byte transaction numbers are used by default, which avoids potential unexpected transaction number overflows, or in some cases, encountering error **R6BT_ERR** (745, *6BTRAN* file required).

This feature can be disabled with the following keyword should this be necessary for backward compatibility:

```
COMPATIBILITY REVERT_TO_V6HDR
```

Standalone applications can disable this support by setting the *cth6flg* global variable to any non-zero value.

Default: OFF

COMPATIBILITY REWRITE_KEY_ERROR

```
COMPATIBILITY REWRITE_KEY_ERROR
```

The key assembly routine that processes a record image to extract key segments and compose key values may return an error condition, **SDAT_ERR**, if data is not available (say because of a missing or truncated field), or if some other error code if a problem arises such as a lack of Unicode support for a Unicode key. The add record routines properly return an error if a key assembly error occurs, but the rewrite routines simply treated the key error equivalent to a NUL or missing key. This is not expected to be a common occurrence since it generally takes a badly formed record image. The rewrite routines now return an error on a failed key assembly by default. This keyword reverts to the prior behavior.

COMPATIBILITY SETEXCABT

```
COMPATIBILITY  SETEXCABT
```

When a node splits that contains key-level locks, the key-level locks must be updated for the old node and the new node. This affects the abort node list. There was a loop that repeatedly attempted to update the abort node list, once for each key-level lock in the node. However, this can be accomplished in a single call. This reduces the contention on the mutex that controls the abort node list. This keyword reverts to the previous approach for testing.

Note: This is intended for internal FairCom testing only.

COMPATIBILITY SPCMGT_INDEX

```
COMPATIBILITY SPCMGT_INDEX
```

Forces c-treeACE to disable the reclamation of unused space management indices with *VLENGTH*, *TRNLOG* data files. Both newly-created variable-length data files and existing variable-length data files will be affected.

COMPATIBILITY STREAM_FILES

`COMPATIBILITY STREAM_FILES`

Forces c-treeACE to ignore the `LOCAL_DIRECTORY` configuration for stream files.

COMPATIBILITY SYNC_LOG

`COMPATIBILITY SYNC_LOG`

DEPRECATED

As of c-treeACE Version 9.0, `COMPATIBILITY LOG_WRITETHRU` is synonymous on both Windows and Unix platforms.

On Unix systems, instructs the c-treeACE Server to open its transaction logs in synchronous write (direct I/O on Solaris) mode. In this mode, writes to the transaction logs go directly to disk (or disk cache), avoiding the file system cache, so the server is able to avoid the overhead of first writing to the file system cache and then flushing the file system cache buffers to disk. This keyword also causes flushed writes for data and index files to use direct I/O. Using this keyword enhances performance of transaction log writes.

Default: OFF

See Also

`COMPATIBILITY LOG_WRITETHRU (page 214)`

COMPATIBILITY TCPIP_CHECK_DEAD_CLIENTS

`COMPATIBILITY TCPIP_CHECK_DEAD_CLIENTS`

With the `COMPATIBILITY TCPIP_CHECK_DEAD_CLIENTS` keyword in the Server configuration, the c-treeACE Server detects when a TCP/IP client has dropped. Every 120 seconds, the client connection socket is rechecked to ensure that it is still a valid communications channel. If a connection is found to be invalid, the Server terminates the connection. This functionality is not currently supported on the c-treeACE Server for the Mac, OS/2, or early Linux (kernel earlier than v2.036) platforms.

The c-treeACE Server normally recognizes when a client disconnects. However, the Server relies on a chain of events controlled by the operating system in order to recognize the disconnection. The client computer must notify the Server host computer that the connection has been dropped. For example: When a user closes an application, the socket is closed by the operating system, which sends a message to the Server host machine. However, if the network connection is temporarily interrupted or if the client machine is powered down suddenly, this message is not sent and the Server host machine can’t recognize that the client connection has dropped.

Default: No check

See Also:

- SESSION_TIMEOUT under "Miscellaneous Control"
- DEAD_CLIENT_INTERVAL (page 175)

COMPATIBILITY TEMP_INDEX_ERROR

`COMPATIBILITY TEMP_INDEX_ERROR`

Consider the following case:

1. Client #1 creates a temporary index on field F.
2. Client #2 changes the value of field F from aaa to zzz.
3. Client #1 changes the value of field F from zzz back to aaa. The record update call fails with error **KDUP_ERR** because the key value already exists in the temporary index.

c-treeACE will ignore a **KDUP_ERR** on a temp index and set *sysiocod* to **IDUP_COD** (-837). Previous behavior can be restored by specifying `COMPATIBILITY TEMP_INDEX_ERROR` in the configuration file.

Note: This keyword also affects temporary index behavior in the case of **ITIM_ERR** on record reads and **KDEL_ERR** on record deletes.

COMPATIBILITY USE_CHARUPPER

`COMPATIBILITY USE_CHARUPPER`

Changes the default method of executing **toupper()** by optionally using **CharUpper()** on the Windows operating system.

COMPATIBILITY V24LOGON

`COMPATIBILITY V24LOGON`

Legacy option that controls whether or not to use old logon connection approach (in which a communication thread performs connection and initial hand- shake communications before launching client thread),

COMPATIBILITY VDLFLG

`COMPATIBILITY VDLFLG`

A **VDLFLG_ERR** indicates that a space-management index entry does not point to available space in a variable-length file. Instead of simply reporting the error (and abandoning a **NEWVREC()** operation), improved default behavior is to remove the space-management index entry, and then continue with **NEWVREC()** processing. The old behavior can be re-established with this option.

11.17 Advanced Configuration Keywords

The c-treeACE Server Configuration File gives the Administrator much more control over the operation of the c-treeACE Server than covered so far. The following detailed list of additional keywords is provided with explanations and default values for each. For completeness, all configuration options supported by the c-treeACE Server are included in this manual. The keywords in this section are listed in alphabetical order.

Note: The options listed in this section require in-depth knowledge of the operating system and hardware configurations of the specific computing environment the c-treeACE process is operating in and, as such, are intended for the appropriate personnel as required. We assume the c-treeACE Administrator will coordinate with the application developer and other members of their operations team as needed.

CONTEXT_HASH (page 331)

Overrides the default number of hash bins for each user.

CRITICAL_SECTION_SPIN (page 332)

Specifies the spin count for c-treeACE critical sections on Windows.

DH_THREAD_STACK_SZ_KB (page 332)

Sets the thread stack size in kilobytes on Unix/Linux (non-Windows) systems.

DIST_COUNT_SEC (page 332)

Sets the time interval between calls to the aggregating functions.

LATCH_SLEEP (page 332)

Specifies the sleep time for a thread synchronization object on Solaris systems (latch). Requires a custom build.

LATCH_SPIN (page 333)

Specifies the spin count for a thread synchronization object on Solaris systems (latch). Requires a custom build.

LOCK_HASH (page 333)

Specifies the number of hash bins available to the lock hash algorithm.

MAX_FILE_WAIT_SECS (page 334)

Sets the number of seconds a user thread waits for the internal threads to close the file before retrying.

MAX_HANDLES (page 334)

Specifies the maximum number of file handles to store keys that do not fit in memory.

MAX_K_TO_USE (page 335)

Specifies the value in kilobytes that can be used for storing key values in memory.

NODE_DELAY (page 335)

Reuses empty index nodes after the specified number of seconds.

NODEQ_SEARCH (page 335)

Specifies how deep (the number of nodes) to inspect the delete node queue for previously deleted index nodes before adding already existing deleted nodes to the queue.

NONTRAN_DATA_FLUSH_SEC (page 336)

Sets the time limit in seconds that a data cache page can remain dirty before it is written to the file system cache.

NONTRAN_INDEX_FLUSH_SEC (page 336)

Sets the time limit in seconds that an index buffer can remain dirty before it is written to the file system cache.

PARTITION_ESTIMATE_LIMIT (page 336)

Estimates the distinct key count for a partitioned index file instead of calculating the exact distinct key count by summing the distinct key counts for each partition of that index.

PREIMAGE_HASH (page 336)

Specifies the number of hash bins available to the preimage hash algorithm.

PRESYNC_THRESHOLD (page 337)

Enables a feature to perform a checkpoint file system flush without blocking all of the mutexes.

SERVER_DIRECTORY (page 337)

Deprecated as of c-treeACE V9.3. See `LOCAL_DIRECTORY` (page 170).

SESSCHG_ENABLE (page 339)

OEM-specific abilities to change the number of sessions under program control. Not for general use.

SETENV (page 339)

Can be used to limit JVM memory.

SKIP_CTADDWORK (page 339)

Disables internal thread synchronization (add work/remove work) logic in early V9 server lines. Consult with the FairCom engineering team.

SYNC_DELAY (page 340)

Specifies the number of seconds between log flushes. FairCom does NOT recommend the use of this option.

TASKER_SLEEP (page 340)

Reduces c-treeACE Server CPU activity level in non-preemptive environments, thus controlling when to put itself to sleep and when to wake up.

TRAN_DATA_FLUSH_SEC (page 340)

Sets the time limit in seconds that a data cache page can remain dirty before it is written to the file system cache.

TRAN_INDEX_FLUSH_SEC (page 340)

Sets the time limit in seconds that an index buffer can remain dirty before it is written to the file system cache.

UDEFER_64YIELD_USEC (page 341)

Specifies the microsecond duration of 64 consecutive yield calls. See the full explanation under `UDEFER_THRESHOLD_USEC` (page 341).

UDEFER_THRESHOLD_USEC (page 341)

Specifies in microseconds the value below which yield calls are used.

VLEN_ERR_RETRY_LIMIT (page 341)

Specifies the number of times to retry an ISAM add or update operation that fails with error **VLEN_ERR**. This option is disabled at compile-time by default.

CONTEXT_HASH

```
CONTEXT_HASH  <# of hash bins>
```

To speed the search for location of ISAM contexts, a simple hashing scheme is used. The number of hash bins defaults to six (6) for each user. If a large number of contexts are to be maintained, then this default can be overridden with this keyword.

Refer to the **OpenISAMContext()**, **SelectISAMContext()**, and **CloseISAMContext()** API calls for detailed information about this feature.

Default: 6

CRITICAL_SECTION_SPIN

```
CRITICAL_SECTION_SPIN <spin_limit>
```

Specify the spin count for c-treeACE critical sections on Windows. Only Windows versions supporting the **CriticalSectionAndSpinCount()** function (0x403 or greater) have this feature enabled.

Default: 1000

DH_THREAD_STACK_SZ_KB

```
DH_THREAD_STACK_SZ_KB=<size>
```

`DH_THREAD_STACK_SZ_KB` sets the thread stack size in kilobytes on Unix/Linux (non-Windows) systems. Each operating system defaults to its own stack size. The system administrator should determine this value from the OS documentation should the value be required.

The default thread stack size for the c-treeACE ISAM server on Unix systems is now 64 KB. The default thread stack size for c-treeACE SQL is 1.5 MB on SCO Unix and 1 MB on other Unix systems. The default thread stack size is 1 MB for both c-treeACE SQL and c-treeACE ISAM Servers on Windows systems.

At server startup, c-treeACE logs the following message to *CTSTATUS.FCS*, to identify the thread stack size that the server is using on Unix systems.

```
Set thread stack size to <stack_size>
```

Note: This keyword only applies to c-treeACE Servers on non-Windows (Unix/Linux) systems.

DIST_COUNT_SEC

```
DIST_COUNT_SEC  <seconds>
```

Sets the time interval between calls to the aggregating functions.

An internal c-treeACE thread, is launched at server startup (if any of the distributed counter features have been enabled). It periodically calls these aggregating functions.

Default: 30 seconds.

LATCH_SLEEP

```
LATCH_SLEEP <latch_sleep_time_in_microseconds>
```

Support for a thread synchronization object on Solaris systems, the latch, can be used as an alternative to a mutex.

See Also

- LATCH_SPIN (page 333)

Note: Latch support is not enabled by default and requires a custom build. Please contact your nearest FairCom office for current availability.

LATCH_SPIN

```
LATCH_SPIN  <latch_spin_count>
```

Support for a thread synchronization object on Solaris systems, the latch, can be used as an alternative to a mutex.

A latch is implemented using the atomic processor instructions **atomic_swap_8()** (which performs an atomic 'test-and-set' operation) and **atomic_and_8()** (which performs an atomic bitwise AND of the specified two values). **set_latch()** calls **atomic_swap_8()** and checks the return value, which is the original value of the latch. If the original value of the latch was zero, this indicates that the thread successfully set the value to one (i.e., acquired the latch). Otherwise, another thread acquired the latch, so the thread retries the operation up to the number of times specified for the latch spin count. If the thread exhausts its spin count without acquiring the latch, the thread sleeps for the number of microseconds specified for the latch sleep time and then retries the operation.

See Also

- LATCH_SLEEP (page 332)

Note: Latch support is not enabled by default and requires a custom build. Please contact your nearest FairCom office for current availability.

LOCK_HASH

```
LOCK_HASH <Number of Hash Bins>
```

The number of hash bins available to the lock hash algorithm. A lock table exists holding all lock entries for each user. To search this table a hash algorithm is employed. The `LOCK_HASH` value specifies the number of 8 byte hash bins available for use by this algorithm. This value should only be increased with careful consideration. There is a marginally decreasing return for increasing values.

Default: 16

The following keywords are closely related the the lock hash bin strategy and are listed here for convenience.

```
LOCK_HASH_MAX <maximum number of lock hash bins>
```

Default: 131072

```
LOCK_HASH_LOADFAC <avg locks per bin>
```

Default: 8

```
LOCK_HASH_REHASHFAC <fraction of hash bins reorganized>
```

Default: 4 (denominator - one fourth at a time)

Note: Dynamic hashing is recommended for lock table entries held before commit. This results in more efficient searching of intermediate lock table contents. Dynamic hashing automatically adjusts to changing conditions of transaction and lock data.

MAX_FILE_WAIT_SECS

`MAX_FILE_WAIT_SECS <seconds> | -1 | 0`

To control delete node and space reclamation threads' access to files, c-tree Server automatically detects when a file open fails due to the internal threads having the file open and it signals the threads to close the file. It then tries to open the file again.

This keyword sets the number of seconds a user thread waits for the internal threads to close the file before retrying.

A value of 0 (the default) sets no limit on the wait loop.

A setting of -1 disables this feature.

The maximum value is 86400. Specifying a greater value sets the limit to 86400 seconds.

This keyword allows an application to avoid error **12** with **sysiocod -8** when trying to open c-tree files in exclusive mode.

Default: 0

MAX_HANDLES

`MAX_HANDLES`

`MAX_HANDLES` can improve the performance of a rebuild.

The rebuild function does a scan on the data file. At the end of the scan the number of records (max key values) is known. An output buffer is used to write sorted key values to disk when needed, that is, when all keys can't fit into data buffers. Data buffers contain unsorted key values put there the data file is scanned. When all data buffers or the pointer area are full, the logic sorts the pointers in the buffer by key value and then writes all data in data buffers to disk. Output is done in output buffer sized chunks to sort work files, which are c-tree data files.

Based on the actual number of keys (number of records) and key size + pntr size the fraction of key values that can stored in the given memory (`MAX_K_TO_USE`) is known. If the memory is large enough then just that amount of memory is needed. If the memory is not large enough then the # of file handles needed to store the values on disk is computed, limited by the `MAX_HANDLES` value.

Default: 255

See Also

`MAX_K_TO_USE (page 335)`

MAX_K_TO_USE

MAX_K_TO_USE

MAX_K_TO_USE can improve the performance of a rebuild. The value is specified in kilobytes (KB) with no scaling factor (such as KB, MB, GB, etc.). A scaling factor is ignored if specified; therefore specifying "4 GB" is the same as "4" and is interpreted as 4 KB. (The SORT_MEMORY keyword supports scaling factors.)

The rebuild function does a scan on the data file. At the end of the scan the number of records (max key values) is known. An output buffer is used to write sorted key values to disk when needed, that is, when all keys can't fit into data buffers. Data buffers contain unsorted key values put there while the data file is scanned. When all data buffers or the pointer area are full, the logic sorts the pointers in the buffer by key value and then writes all data in data buffers to disk. Output is done in output buffer sized chunks to sort work files, which are c-tree data files.

Based on the actual number of keys (number of records) and key size + pntr size the fraction of key values that can stored in the given memory (MAX_K_TO_USE) is known. If the memory is large enough then just that amount of memory is needed. If the memory is not large enough then the number of file handles needed to store the values on disk is computed, limited by the MAX_HANDLES (, http://docs.faircom.com/doc/ctserver/#52143.htm) value.

> If both SORT_MEMORY and MAX_K_TO_USE are specified in *ctsrvr.cfg*, only the one that is specified *last* in the configuration file takes effect.

Default: 10000 (10 MB)

See Also

SORT_MEMORY (page 202)
MAX_HANDLES (page 334)

NODE_DELAY

NODE_DELAY <seconds>

Reuses empty index nodes after NODE_DELAY seconds. This allows the c-treeACE Server to finish active searches in the index tree before removing the empty node.

Default: 300

NODEQ_SEARCH

NODEQ SEARCH <nodes>

Inspects the delete node queue *<nodes>* deep for previously deleted index nodes before adding already existing deleted nodes to the queue.

Default: 50

In server implementations, an emptied index node is not directly deleted from the b-tree. Instead, an entry is placed in a queue read by the special administrative delete-node-thread. Before writing to the queue, a check is made to see if the node is already on the queue. For systems that generate many empty nodes, it can be more efficient to inspect the queue to a deeper level.

NONTRAN_DATA_FLUSH_SEC

```
NONTRAN_DATA_FLUSH_SEC   <time_limit_in_seconds>
```

Sets the time limit in seconds that a data cache page can remain dirty before it is written to the file system cache.

- Specify `IMMEDIATE` to cause dirty pages to be written immediately.
- Specify `OFF` to disable time limit-based flushing.

Default: 60 seconds

See Also

- Controls for Performance AND Safety of Non-Transaction Updates (page 128)

NONTRAN_INDEX_FLUSH_SEC

```
NONTRAN_INDEX_FLUSH_SEC  <time_limit_in_seconds>
```

Sets the time limit in seconds that an index buffer can remain dirty before it is written to the file system cache.

- Specify `IMMEDIATE` to cause dirty pages to be written immediately.
- Specify `OFF` to disable time limit-based flushing.

Default: 60 seconds

See Also

- Controls for Performance AND Safety of Non-Transaction Updates (page 128)

PARTITION_ESTIMATE_LIMIT

```
PARTITION_ESTIMATE_LIMIT  <limit>
```

This keyword estimates the distinct key count for a partitioned index file instead of calculating the exact distinct key count by summing the distinct key counts for each partition of that index. This behavior can significantly reduce the time necessary to calculate the distinct key count for a partitioned index (at the cost of a larger uncertainty in the estimate).

Default sampling behavior is to only sample three active partitions unless `PARTITION_ESTIMATE_LIMIT` sets this limit to a value greater than three. The partitions sampled are the first and last partitions that ordinarily would have been used, and one or more in the "middle" of the remaining active partitions. At this time no attempt is made to spread out the sampled middle partitions over the range of available middle partitions.

With a negative value, the behavior resorts to summing the distinct key counts for each active partition.

PREIMAGE_HASH

```
PREIMAGE_HASH <number of hash bins>
```

The number of hash bins available to the preimage hash algorithm. During a transaction, a preimage space is used to hold intermediate results pending an abort or a commit. To search this area a hashing algorithm is employed. The `PREIMAGE_HASH` value specifies the number of four

byte bins available per user for use by this algorithm. This value should be increased only if a large volume of updates and/or additions per transaction (e.g., several thousand) is anticipated.

Default: 128

The following keywords are closely related the the lock hash bin strategy and are enumerated here for convenience.

`PREIMAGE_HASH_MAX <maximum number of lock hash bins>`

Default: 131072

`PREIMAGE_HASH_LOADFAC <avg locks per bin>`

Default: 8

`PREIMAGE_HASH_REHASHFAC <fraction of hash bins reorganized>`

Default: 4 (denominator - one fourth at a time)

Note: Dynamic hashing is now used for transaction entries held before commit. This results in more efficient searching of intermediate transaction contents. Dynamic hashing automatically adjusts to changing conditions of transaction and lock data.

PRESYNC_THRESHOLD

`PRESYNC_THRESHOLD <active user count>`

Background: When a transaction is committed, the transaction log must be flushed to disk to ensure the log contains the ability to undo or redo the transaction. The server does not, however, flush all the transaction related data and index file images to disk. The c-tree cache pages holding the disk images are aged and eventually forced to disk, otherwise transaction logs of arbitrary age would be required to make good any transaction that was committed but not entirely flushed to disk. The aging of cache pages occurs as part of the checkpoint, and consists of issuing an OS write operation. If the OS write is not guaranteed to synchronously flush the image to disk, then a call must also be made to flush the system cache pages (as opposed to the c-tree cache pages) to disk. When a data or index disk image for a transaction is committed but not yet flushed to disk, the transaction is placed on a data or index vulnerable transaction list. When the image is flushed to disk, the transaction is removed from the list. A feature can be enabled to perform the checkpoint file system flush without blocking all of the mutexes.

Specifying *<active user count>* greater than or equal to zero enables this feature. An active user count less than zero turns off this feature. The default for the threshold is set to -1 which turns off the new feature. [By active user count we mean the number of users actually making requests to the server. An idle user, or a user in between requests, will not count toward the active user count.]

Pending further testing, it is suggested that the `PRESYNC_THRESHOLD` be set somewhere in the range of 2 to 6.

SERVER_DIRECTORY

`SERVER_DIRECTORY <path>`

See LOCAL_DIRECTORY (page 170)

Note: This configuration option has been deprecated as of c-treeACE V9.3 and later. LOCAL_DIRECTORY (page 170) is now the preferred keyword to allow the server to store data and files in an alternative location.

To avoid potential problems with the use of this option, it has been disabled. When this option is specified in *ctsrvr.cfg*, the c-tree Server fails to start and displays the following message:

```
The SERVER_DIRECTORY option is no longer supported.
Use the LOCAL_DIRECTORY option instead.
```

The message is logged to *CTSTATUS.FCS*. On Unix systems it is also written to standard output and on Windows systems it is displayed in a dialog box when the c-tree Server is not running as a Windows service.

Relocating Transaction Logs

Use these configuration keywords to relocate transaction logs if this has been the purpose of using this keyword in the past:

```
LOG_EVEN (page 219)
LOG_ODD (page 219)
START_EVEN (page 223)
START_ODD (page 224)
```

In V10.3 and later, these configuration options can include an environment variable name that will be substituted with its value when the configuration file is read.

Legacy Notes

The SERVER_DIRECTORY option was one of two mutually exclusive ways to supply the name of a directory path the c-treeACE Server used when processing all files not having absolute names (i.e., absolute names include a specific volume or drive reference as part of the name). For example, *d:\fairserv\data* (the trailing slash was required). The other option, LOCAL_DIRECTORY (page 170), is now the preferred keyword to allow the server to store data and files in an alternative location.

If a SERVER_DIRECTORY name was defined in the configuration script, the name was attached to the beginning of any file name that was not absolute. If no SERVER_DIRECTORY or LOCAL_DIRECTORY name was supplied, all database and system files were stored relative to the c-treeACE Server working directory. SERVER_DIRECTORY and LOCAL_DIRECTORY cannot be used together. SERVER_DIRECTORY did not affect the location of the c-treeACE Server Status log, the transaction log files, or the start files.

Note: The SERVER_DIRECTORY, unlike LOCAL_DIRECTORY, became part of the file name. The name entered into the transaction log included the SERVER_DIRECTORY.

Default: Server working directory

See Also

- LOCAL_DIRECTORY

SESSCHG_ENABLE

```
SESSCHG_ENABLE
```

This option pertains to OEM-specific abilities to change the number of sessions under program control. The internal variable, *ct_mxu1*, which controls memory allocations and for loops was not properly handled when **SESSVAL()** and **SESSINC()** were used. This fix sets *ct_mxu1* to the activation limit when `SESSCHG_ENABLE YES` is added to the server configuration file. If this entry is not in the configuration file, then calls to **SESSVAL()** and **SESSINC()** will return **NSUP_ERR** (454, not supported).

Note: SESSCHG_ENABLE should not be considered for general use.

SETENV

```
SETENV  DH_JVM_OPTION_STRINGS=-Xms100m -Xmx300m
```

Limits JVM memory.

SKIP_CTADDWORK

```
SKIP_CTADDWORK  <YES | NO>
```

Disables internal thread synchronization (add work/remove work) logic in early V9 server lines.

Note: Only consider this option after consultation with the FairCom engineering team.

SUBSYSTEM SQL LATTE

```
SUBSYSTEM SQL LATTE
```

Marks a section in the server configuration file *ctsrvr.cfg* to configure the LATTE sorting subsystem for c-treeACE SQL.

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

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

This subsystem accepts the following keywords:

- `MAX_MEMORY` - Maximum advisable memory per environment.
- `CACHE_BLK` - Default number of blocks in the temporary cache of a table.
- `NONE` - This option is used in conjunction with the tamper-proof settings file (*.set*). When specified, the entire SUBSYSTEM SQL LATTE cannot be overridden in the *ctsrvr.cfg* file.

Example

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

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

SYNC_DELAY

`SYNC_DELAY <seconds>`

Specifies the number of seconds between log flushes.

Note: FairCom does NOT recommend the use of this option since it may cause problems with our automatic recovery.

TRAN_DATA_FLUSH_SEC

`TRAN_DATA_FLUSH_SEC <time_limit_in_seconds>`

In V11 and later, sets the time limit in seconds that a data cache page can remain dirty before it is written to the file system cache.

- Specify `IMMEDIATE` to cause dirty pages to be written immediately.
- Specify `OFF` to disable the time limit-based flushing.

This option can be changed using the **ctSETCFG()** API function or the **ctadmn** utility.

Default: 60

TRAN_INDEX_FLUSH_SEC

`TRAN_INDEX_FLUSH_SEC <time_limit_in_seconds>`

In V11 and later, sets the time limit in seconds that an index buffer can remain dirty before it is written to the file system cache.

- Specify `IMMEDIATE` to cause dirty pages to be written immediately.
- Specify `OFF` to disable the time limit-based flushing.

This option can be changed using the **ctSETCFG()** API function or the **ctadmn** utility.

Default: 60

TASKER_SLEEP

`TASKER_SLEEP <milliseconds | 0 | -1>`

Reduces c-treeACE Server CPU activity level in non-preemptive environments, where the c-treeACE Server performs its own task switching, thus controlling when to put itself to sleep and when to wake up. The value of TASKER_SLEEP controls whether the c-treeACE Server puts itself to sleep and, if it does, when it will wake itself up. Setting this keyword to other than default will diminish the c-treeACE Server performance. This keyword is generally only required in older Unix environments. (SCO Unix, QNX, Apple A/UX, Interactive Unix and Motorola 88 OPEN.)

Valid values are:

Value	Explanation
-1	Never sleep.
0	Relinquish control of the CPU if another process is ready to run; otherwise, resume processing
> 0	Sleep for that amount of time (in milliseconds).

Default: 0

UDEFER_64YIELD_USEC

```
UDEFER_64YIELD_USEC  <defer>
```

where <defer> is the microsecond duration of 64 consecutive yield calls. See the full explanation under UDEFER_THRESHOLD_USEC (page 341).

See Also

- UDEFER_THRESHOLD_USEC (page 341)

UDEFER_THRESHOLD_USEC

```
UDEFER_THRESHOLD_USEC
```

An internal defer routine takes a microsecond sleep argument. However, not all systems provide a useful implementation of a microsecond or nanosecond sleep routine. The UDEFER_THRESHOLD_USEC and UDEFER_64YIELD_USEC (page 341) configuration options force this routine to make multiple calls to **defer(0)**, which causes a thread yield, to generate shorter (more accurate) sleep intervals.

```
UDEFER_THRESHOLD_USEC <usec value below which yield calls used>
UDEFER_64YIELD_USEC   <usec duration of 64 consecutive yield calls>
```

Even if the microsecond sleep has provisions for consecutive yields, it will not take effect unless both the DEFER_THRESHOLD_USEC and UDEFER_64YIELD_USEC appear with non-zero values.

See Also

- UDEFER_64YIELD_USEC (page 341)

VLEN_ERR_RETRY_LIMIT

```
VLEN_ERR_RETRY_LIMIT <limit>
```

`VLEN_ERR_RETRY_LIMIT` is used to set a number of times to retry an ISAM add or update operation that fails with error **VLEN_ERR**. Under normal operation, a **VLEN_ERR** error (148, **WRTVREC()** cannot fit record at *recbyt*) is unexpected.

Note: This option is disabled at compile-time by default.

11.18 Diagnostics Keywords

Diagnostic keywords are used to provide additional detailed c-treeACE monitoring of specific internal parameters and metrics. Typically, these keywords should only be used under the specific advice of a FairCom engineer as they may negatively impact the performance of your c-treeACE Server. Please do not hesitate to contact your nearest FairCom office should you have any questions regarding use of any of these keywords.

WARNING: The keywords listed below should be used ONLY on the advice of your application developer. They can seriously alter the operation of the c-treeACE Server.

DIAGNOSTIC_INT (page 347)

Overrides the size of the default internal buffer for trapping the communications is 128K bytes. Should only be used under advice from a FairCom engineer.

DIAGNOSTIC_STR (page 347)

Prepends a path onto the name of the communications trap file (*TRAPCOMM.FCS*). Should only be used under advice from a FairCom engineer.

DIAGNOSTICS ABEND_ABORT (page 347)

Enables a method for generating a process core on an abnormal shutdown by having c-treeACE call **abort()**.

DIAGNOSTICS ABORT_NODE_LIST (page 347)

Enables a circular memory buffer to hold information relevant to debugging a L59 error.

DIAGNOSTICS AUTO_PREIMG_CHECKLOCK / AUTO_PREIMG_CHECKREAD (page 348)

Causes the file mode for files affected by `AUTO_PREIMG` configuration entries to be augmented by one or both of *ctCHECKLOCK* and *ctCHECKREAD*. Not for changing the file mode stored on disk for the files in question.

DIAGNOSTICS AUTO_TRNLOG_CHECKLOCK / AUTO_TRNLOG_CHECKREAD (page 348)

Causes the file mode for files affected by the `AUTO_TRNLOG` configuration entry will be augmented by one or both of *ctCHECKLOCK* and *ctCHECKREAD*. Not for changing the file mode stored on disk for the files in question.

DIAGNOSTICS CHECK_UDEFER (page 349)

Used to tune UDEFER options. Internal FairCom engineering use only - Not for production use.

DIAGNOSTICS COMM_LEVEL_X (page 349)

Enables communication diagnostics.

DIAGNOSTICS DBGSEMTIM (page 349)

Enables high-resolution measurement of the time threads wait at each mutex and semaphore call.

DIAGNOSTICS DEBUG (page 349)

Enables a general method to signal debugging intent and to input data for diagnostic or debugging use.

DIAGNOSTICS EXTENDED_TRAN_NO (page 349)

Forces the server to log each physical open of a non-extended transaction number file to the *CTSTATUS.FCS* file.

DIAGNOSTICS FILE_LOGON (page 350)

Causes the following four file counters to be output upon each logon and logoff: physical files open, logical files open, File Control Blocks (FCBs) in use, and FCBs available.

DIAGNOSTICS FLUSH_BLM (page 350)

Enables at file close, a check to see if any buffer or cache pages have been missed when flushing and clearing the buffer/cache space.

DIAGNOSTICS FORCEI_SHADOWUPD (page 350)

Enables a check for unexpected instances of shadow updates during shadow entry clean-up and will list such updates in a server's *CTSTATUS.FCS*.

DIAGNOSTICS KEY_COMPARE (page 350)

Enables an index tree-walk history which contains the nodes visited as well as the key comparisons made.

DIAGNOSTICS KLLX (page 350)

Enables, at the end of automatic recovery, all leaf nodes to be checked to see if any uncommitted key values were inadvertently treated as committed.

DIAGNOSTICS L59 (page 350)

Reverts server to the prior behavior of a server termination with an L59 failure code when flaws are detected in abort node list processing

DIAGNOSTICS LOCK_LOGON (page 351)

Displays the net lock count upon each c-treeACE Server logon and logoff.

DIAGNOSTICS LOGON_COMM (page 351)

Enables the logon communications sequence to be output by the server.

DIAGNOSTICS LOWL_CRC_ON (page 351)

Enables low-level CRC communication checks.

DIAGNOSTICS MEMORY_LEAK (page 351)

Enables debug information upon each memory **alloc()** (get) and **free()** (put) output to *MEMLEAK.FCS*. This option will result in extreme performance degradation.

DIAGNOSTICS MEMTRACK (page 351)

Enables the c-treeACE Server memory tracking feature. Should be used only when recommended by a FairCom engineering team member - It will impact performance.

DIAGNOSTICS NO_EXCEPTION_HANDLER (page 353)

Disables the Exception Handler in case the addition of this error handler created any unexpected behavior.

DIAGNOSTICS NO_LOG_EXTENSION (page 353)

Forces the transaction log extension logic to skip the writing of 0xff fill. The logs are then extended only as actual log writes take place.

DIAGNOSTICS NODE_REQUEST_TIME (page 353)

Enables tracking the cumulative time spent finding each index node in the index cache.

DIAGNOSTICS NONE (page 354)

Explicitly blocks certain keywords from being used in a subsequent stage of configuration loading.

DIAGNOSTICS PCRP_ERR (page 354)

Enables the occurrence of a **PCRP_ERR** to log a message to *CTSTATUS.FCS* and create a process stack trace or minidump of the process. Requires a custom build.

DIAGNOSTICS PROCESS_EXIT (page 354)

Enables c-treeACE to take the following actions before exiting: 1) Dump a process stack trace. 2) Display a dialog box on Windows.

DIAGNOSTICS QUEUE_LOGON (page 355)

Provides the current number of items in the c-treeACE Server queues.

DIAGNOSTICS REMAINING_THREADS (page 356)

Enables listing threads still active at server shutdown (causing the final system checkpoint to be skipped) to the *CTSTATUS.FCS* log file.

DIAGNOSTICS REPL_READ_BUFFER (page 356)

Enables a check that the log data is being correctly read into the replication buffer.

DIAGNOSTICS SUBALLOCATOR_OFF (page 356)

Forces c-treeACE to allocate memory directly from the heap instead of using its internal memory suballocator.

DIAGNOSTICS THREAD_DUMP (page 356)

Enables low-level thread diagnostics.

DIAGNOSTICS TRACK_LOGON (page 356)

Provides a net count of memory allocation requests. See also `MEMORY_TRACK`.

DIAGNOSTICS TRAN_RECOVERY (page 357)

Causes transaction recovery log messages to be written to the file *RECOVERY.FCS.*

DIAGNOSTICS TREE_WALK (page 357)

Enables an index tree-walk history which contains the nodes visited as well as the key comparisons made.

DIAGNOSTICS UPDFLG (page 357)

Logs each change to a file update flag (*updflg*) to *CTSTATUS.FCS*.

DIAGNOSTICS WRITE_ERR_DUMP (page 357)

Causes the contents of a write request to be appended to the *WRITE_ERR.FCS* file and a notification to be logged to *CTSTATUS.FCS* if a **WRITE_ERR** occurs.

DIAGNOSTICS WRITETHRU (page 358)

Causes all file names to be listed in the *CTSTATUS.FCS* file if it detects a file that is not under transaction control and does not have *WRITETHRU* defined.

DIAGNOSTIC_INT

The default internal buffer for trapping the communications is 128K bytes. An entry of the following form will override the default buffer size.

```
DIAGNOSTIC_INT  <buffer size in K bytes>
```

Each time the buffer fills, it is dumped to the trap file. For example,

```
DIAGNOSTIC_INT  48
```

would create a 48K byte buffer. The name of the trap file is derived from the communication protocol name by adding *.FCS*.

Note: Any number of `DIAGNOSTIC_INT` entries can be included in the configuration file. These create an internal dynamically allocated array used for providing input to various debugging routines. These options should only be used under advice from a FairCom engineer.

DIAGNOSTIC_STR

By default, the name of the communications trap file is *TRAPCOMM.FCS* and will be located in the server directory. To prepend a path onto the trap file name (say to route it to a separate disk), add an entry of the form

```
DIAGNOSTIC_STR  <trap file path>
```

For example, to append the file system name */bigdisk/* to the filename, include the following entry in the configuration file:

```
DIAGNOSTIC_STR  /bigdisk/
```

The trap file would be then be located in */bigdisk/TRAPCOMM.FCS*

Note: Any number of `DIAGNOSTIC_STR` entries can be included in the configuration file. These create an internal dynamically allocated array used for providing input to various debugging routines. Additional entries should only be used under advice from a FairCom engineer.

DIAGNOSTICS ABEND_ABORT

```
DIAGNOSTICS ABEND_ABORT
```

Enables a method for generating a process core on an abnormal shutdown by having c-treeACE call **abort()**. This does cause the attempt at closing files and cleaning up to be skipped, so recovery may take a longer time to run when this keyword is specified.

DIAGNOSTICS ABORT_NODE_LIST

```
DIAGNOSTICS ABORT_NODE_LIST
```

Enables a circular memory buffer to hold information relevant to debugging a L59 error. In the event of a L59 failure, the memory log will be dumped to the text file *ABNODLST.FCS*. In particular, placing in the server configuration file turns on the memory log. Since it is a circular buffer, only the most recent entries are maintained. Each entry requires 36 bytes.

The default number entries can be overridden through the configuration entry

```
DIAGNOSTIC_INT  <override value>
```

If the L59 occurs only in a well defined number of indices, then the DIAGNOSTIC_STR key word can also be used to list the relevant index files. For example, the following server configuration entries would cause the memory log to hold 20,000 entries for the listed indices:

```
DIAGNOSTICS     ABORT_NODE_LIST
DIAGNOSTIC_INT  20000
DIAGNOSTIC_STR  customer.idx
DIAGNOSTIC_STR  invoice.idx
DIAGNOSTIC_STR  product.idx
```

See Also

- DIAGNOSTIC_INT (page 347)
- DIAGNOSTIC_STR (page 347)

DIAGNOSTICS AUTO_PREIMG_CHECKLOCK / AUTO_PREIMG_CHECKREAD

```
DIAGNOSTICS AUTO_PREIMG_CHECKLOCK
DIAGNOSTICS AUTO_PREIMG_CHECKREAD
```

When either of these diagnostic options are used, the file mode for files affected by AUTO_PREIMG configuration entries will be augmented by one or both of *ctCHECKLOCK* and *ctCHECKREAD*. Missing lock calls will then generate **DADV_ERR** (57) errors.

Note: It is not the intention of these diagnostic options to change the file mode stored on disk for the files in question. The *ctCHECKLOCK* and/or *ctCHECKREAD* modes should not be added to the files' header images.

See Also

```
AUTO_PREIMG (page 209)
AUTO_TRNLOG (page 209)
AUTO_TRNLOG_LIGHT (page 210)
DIAGNOSTICS AUTO_TRNLOG_CHECKLOCK (page 348)
DIAGNOSTICS AUTO_TRNLOG_CHECKREAD (page 348)
PREIMAGE_DUMP (page 280)
```

DIAGNOSTICS AUTO_TRNLOG_CHECKLOCK / AUTO_TRNLOG_CHECKREAD

```
DIAGNOSTICS AUTO_TRNLOG_CHECKLOCK
DIAGNOSTICS AUTO_TRNLOG_CHECKREAD
```

When either of these diagnostic options are used, the file mode for files affected by the AUTO_TRNLOG configuration entry will be augmented by one or both of *ctCHECKLOCK* and *ctCHECKREAD*. Missing lock calls will then generate **DADV_ERR** (57) errors.

Note: It is not the intention of these diagnostic options to change the file mode stored on disk for the files in question. The *ctCHECKLOCK* and/or *ctCHECKREAD* modes should not be added to the files' header images.

See Also

```
AUTO_PREIMG (page 209)
```

AUTO_TRNLOG (page 209)
AUTO_TRNLOG_LIGHT (page 210)
DIAGNOSTICS AUTO_PREIMG_CHECKLOCK (page 348)
DIAGNOSTICS AUTO_PREIMG_CHECKREAD (page 348)
PREIMAGE_DUMP (page 280)

DIAGNOSTICS CHECK_UDEFER

DIAGNOSTICS CHECK_UDEFER

Used to tune UDEFER options.

Note: Internal FairCom engineering use only. Not intended for production use.

DIAGNOSTICS COMM_LEVEL_X

DIAGNOSTCIS COMM_LEVEL_X

Enables communication diagnostics. Valid level:

COMM_LEVEL_1

DIAGNOSTICS DBGSEMTIM

DIAGNOSTICS DBGSEMTIM

Enables high-resolution measurement of the time threads wait at each mutex and semaphore call.

DIAGNOSTICS DEBUG

DIAGNOSTICS DEBUG

Enables a general method to signal debugging intent and to input data for diagnostic or debugging use.

DIAGNOSTICS EXTENDED_TRAN_NO

DIAGNOSTICS EXTENDED_TRAN_NO

This keyword forces the server to log each physical open of a non-extended transaction number file to the *CTSTATUS.FCS* file. 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 exceed the original 4-byte range and one of these files is updated. By “all files” we mean superfile hosts and indices; data files are not affected by the extended transaction number attribute.

See Also

COMPATIBILITY 6BTRAN_NOT_DEFAULT (page 305)
COMPATIBILITY EXTENDED_TRAN_ONLY (page 309)

DIAGNOSTICS FILE_LOGON

```
DIAGNOSTICS FILE_LOGON
```

When the `DIAGNOSTICS FILE_LOGON` keyword is added to the c-treeACE Server configuration file, upon each logon and logoff, four file counters are output: physical files open, logical files open, File Control Blocks (FCBs) in use, and FCBs available. The logical files count will be greater than physical files if c-tree Superfiles are in use. FCBs in use count will be greater than logical files if index files contain additional index members. These values reflect counts generated by all applications using the c-treeACE Server.

Default: OFF

DIAGNOSTICS FLUSH_BLM

```
DIAGNOSTICS FLUSH_BLM
```

Enables at file close, a check to see if any buffer or cache pages have been missed when flushing and clearing the buffer/cache space. If a page is missed then *CTSTATUS.FCS* will contain one or messages beginning with the line:

```
DIAGNOSTIC: orphaned BLM ...
```

DIAGNOSTICS FORCEI_SHADOWUPD

```
DIAGNOSTICS FORCEI_SHADOWUPD
```

Enables a check for unexpected instances of shadow updates during shadow entry clean-up and will list such updates in a server's *CTSTATUS.FCS*.

DIAGNOSTICS KEY_COMPARE

```
DIAGNOSTICS     KEY_COMPARE
```

Enables an index tree-walk history which contains the nodes visited as well as the key comparisons made. Each time a new tree-walk is begun, the history is refreshed. If a terr() occurs, the the history is dumped into the file *LOGTREE.FCS*. This information is not maintained during high speed index loads, or during cleaning transaction information from an index.

DIAGNOSTICS KLLX

```
DIAGNOSTICS KLLX
```

Enables, at the end of automatic recovery, all leaf nodes to be checked to see if any uncommitted key values were inadvertently treated as committed. If so, the server console displays the index name and the suspect transaction number.

DIAGNOSTICS L59

```
DIAGNOSTICS L59
```

Prior to V8, logic to detect flaws in abort node list processing generated an L59 failure with server termination. New logic post V8 will mark the index file bad and close the low level physical fail failing with error **BIDX_ERR** (527) and output the following message to *CTSTATUS.FCS*:

```
"Trouble processing new index node"
```

This keyword reverts server to the prior behavior of a server termination with an L59 failure code.

DIAGNOSTICS LOCK_LOGON

```
DIAGNOSTICS LOCK_LOGON
```

The `DIAGNOSTICS LOCK_LOGON` keyword displays the net lock count upon each c-treeACE Server logon and logoff. This count is system wide, not just for the process logging off or on and incurs very low overhead.

Default: OFF

DIAGNOSTICS LOGON_COMM

```
DIAGNOSTICS LOGON_COMM
```

(Legacy) Enables the logon communications sequence to be output by the server.

DIAGNOSTICS LOWL_CRC_ON

```
DIAGNOSTICS LOWL_CRC_ON
```

Enables low-level CRC communication checks.

DIAGNOSTICS MEMORY_LEAK

```
DIAGNOSTICS MEMORY_LEAK
```

Enables debug information upon each memory **alloc()** (get) and **free()** (put) output to *MEMLEAK.FCS*. A typical line of output looks like:

```
003b860cx 0000070 getmem O01 B0026 72
```

The line is interpreted as follows:

```
entry  interpretation
-----  --------------
1st    memory address
2nd    sequence number to aid debugging
3rd    action
4th    thread ID
5th    excess of allocations over frees.
         Separate "balance" counts maintained for mballc's and ctgetmem's
6th    memory size (not given for mbfree's)
```

Note: Use of this option will result in extreme performance degradation.

DIAGNOSTICS MEMTRACK

```
DIAGNOSTICS MEMTRACK
```

This configuration option enables the c-treeACE Server memory tracking feature.

`DIAGNOSTICS MEMTRACK` increases memory use for each allocation by the size of the *MEMALC* structure (80 bytes for 32-bit compiles and 152 bytes for 64-bit compiles). The maximum number of stack frames to collect per call stack is set at compile time to *MEMSTKLMT*, which is defined to 15 in *memtrk.c*.

If a client attempts to read or log memory statistics using the **ctMEMSTAT()** API function, however, c-treeACE is running without `DIAGNOSTICS MEMTRACK` in *ctsrvr.cfg*, **ctMEMSTAT()** returns error code **FTYP_ERR** (53, file mode inconsistent with c-tree config).

Note: This option should be used only when recommended by a FairCom engineering team member as it will impact performance, and the collected data is only meaningful for internal diagnostics. It is included here for completeness.

In V11 and later, the memory allocation tracking feature of c-treeACE Server is supported on Linux systems. To use this feature, add `DIAGNOSTICS MEMTRACK` to *ctsrvr.cfg*.

The `MEMTRACK` keyword allows the **ctstat** utility to be used to log current memory allocations to a file. The following parameters can be used with **ctstat**:

- *-mf logfile* - Log all memory allocations to the specified file
- *-ma logfile* - Log aggregate memory allocations to the specified file
- *-mr min,max* - Log only memory allocations in the range min,max
- *-ms* - Output memory allocation statistics

Examples

```
C:\>ctstat -ms -h 10 -s FAIRCOMS
      memseq      memalc
        1267         992
        1289         997
        1289         997
        1289         997
```

To log all memory allocations (with each allocation listed separately) to the file *memfull.log* in the c-treeACE Server's working directory:

```
C:\>ctstat -mf memfull.log -i 1 1 -s FAIRCOMS
```

To log all memory allocations (with allocations having the same call stack listed only once each) to the file *memaggr.log* in the c-treeACE Server's working directory:

```
C:\>ctstat -ma memaggr.log -i 1 1 -s FAIRCOMS
```

To log all memory allocations that have sequence numbers between 1900 and 2000 to the file *memaggr.log* in the c-treeACE Server's working directory:

```
C:\>ctstat -ma memaggr.log -mr 1900,2000 -i 1 1 -s FAIRCOMS
```

See Also

DIAGNOSTICS SUBALLOCATOR_OFF (page 356)

DIAGNOSTICS NO_EXCEPTION_HANDLER

```
DIAGNOSTICS NO_EXCEPTION_HANDLER
```

The c-treeACE Server for Windows includes a Win32 Exception Handler to take care of any error situations. This keyword disables the Exception Handler in case the addition of this error handler created any unexpected behavior. FairCom cannot foresee any situation where the keyword will be needed, but in the interest of safety, added a method for disabling this Exception check.

Default: Handler Enabled

DIAGNOSTICS NO_LOG_EXTENSION

```
DIAGNOSTICS NO_LOG_EXTENSION
```

Forces the transaction log extension logic to skip the writing of 0xff fill. The logs are then extended only as actual log writes take place.

DIAGNOSTICS NODE_REQUEST_TIME

```
DIAGNOSTICS NODE_REQUEST_TIME
```

Enables tracking the cumulative time spent finding each index node in the index cache. The **LockDump()** function includes the node request times in its output. The average and total node request times are reported in milliseconds.

Example

```
Buffer request count profile:
       Avg req time Tot req time Request count Block count  Node offset         Filno Type Filename
       ------------ ------------ ------------- ------------ ------------------ ----- ----
--------------------
NRQ:              3       605087        200000        67078 0x0000000000033e000    26    R
admin_sys_001_000001613.idx
NRQ:              1       298947        200000        33912 0x0000000000008000    21    R
admin_sys_001_000001693.idx
NRQ:              0        13977         94150         1544 0x0000000000033c000    26    I
admin_sys_001_000001613.idx
NRQ:              0         7762         94149          740 0x0000000000000c000    26    I
admin_sys_001_000001613.idx
NRQ:              0          177         11701           11 0x00000000005fc000    26    I
admin_sys_001_000001613.idx
NRQ:              0           64           269            0 0x00000000004f6000    26    L
admin_sys_001_000001613.idx
NRQ:              0           60           269            0 0x00000000003c8000    26    L
admin_sys_001_000001613.idx
NRQ:              0           60           269            0 0x00000000001f6000    26    L
admin_sys_001_000001613.idx
NRQ:              0           57           269            0 0x00000000002b2000    26    L
admin_sys_001_000001613.idx
NRQ:              0           56           269            0 0x00000000004ae000    26    L
admin_sys_001_000001613.idx
```

DIAGNOSTICS NODEQ_MESSAGE

```
DIAGNOSTICS NODEQ_MESSAGE
```

Enables the delete node thread to log the following message to *CTSTATUS.FCS* when it finds a discrepancy between a delete node queue entry and the current index file on disk having the same filename:

```
ctdnode: could not process empty node
```

This message is suppressed by default.

The delete node thread logs the following message to *CTSTATUS.FCS* when it finds a discrepancy between a delete node queue entry and the current index file on disk having the same filename:

```
"ctdnode: could not process empty node"
```

For example, if an index file is deleted and recreated, a delete node queue entry for the original index file will not match the file ID values in the new index file.

As another example, if an index file is blocked using the file block feature, the delete node thread will not be able to access the file. Because this is a normal message, we now suppress it by default.

If desired, the message can be re-enabled by adding the option `DIAGNOSTICS NODEQ_MESSAGE` to *ctsrvr.cfg*.

DIAGNOSTICS NONE

```
DIAGNOSTICS NONE
```

This option is used in conjunction with the tamper-proof settings file under the server. Configuration options that are in the encrypted settings file, *ctsrvr.set*, cannot be overridden in the *ctsrvr.cfg* file.

The DIAGNOSTICS, COMPATIBILITY, and CONSOLE keywords do not automatically block use in a subsequent stage of configuration loading. To explicitly block any of these keywords present in a later stage, add entries in the form: <keyword> NONE where <keyword> is DIAGNOSTICS, COMPATIBILITY, or CONSOLE.

Default: Not present

DIAGNOSTICS PCRP_ERR

```
DIAGNOSTICS PCRP_ERR
```

Enables at any time a **PCRP_ERR** occurs, a message is logged to *CTSTATUS.FCS* and a process stack trace or minidump of the process is created.

Example messages:

```
PCRP_ERR: ctpartno(), file <filename>
PCRP_ERR: ctunfoldpartno(), file <filename> loc <location>
PCRP_ERR: subno=<subno> dnum->ptcur=<ptcur> dnum->ptmbr=<ptmbr>
```

DIAGNOSTICS PROCESS_EXIT

```
DIAGNOSTICS PROCESS_EXIT
```

Enables c-treeACE to take the following actions before exiting (even on a normal shutdown):

1. Dumps a process stack trace, logging the name of the stack dump file to *CTSTATUS.FCS*.
2. Displays one of several dialog boxes on Windows.

Note: This feature requires a custom build to enable.

DIAGNOSTICS QUEUE_LOGON

```
DIAGNOSTICS QUEUE_LOGON
```

The `DIAGNOSTICS QUEUE_LOGON` provides the current number of items in the c-treeACE Server queues. Three system wide counts are given for `QUEUE_LOGON`. These three counts, listed below, are preceded by the letters M, C and D, respectively. This option requires very low overhead.

- The number of pending messages in c-treeACE Server monitors.
- The number of pending messages in checkpoint queue.
- The number of pending messages in the delete node.

Default: OFF

DIAGNOSTICS READ_ERR

Rare, but unexpected **READ_ERR** (36) messages have been reported. When the operating system denies a read request from the file system, normally, a system error code is reported such as "permissions denied" or otherwise. However, rare and sporadic cases have been reported where this additional error return was zero (0). Good practice dictates we should determine why the OS file system is denying a read operation for any reason. A diagnostic option is now available to generate additional context information, along with a server stack trace when this condition occurs.

```
DIAGNOSTICS READ_ERR
```

This option enables additional diagnostic logging of read errors. When this option is in effect and an unexpected read error occurs (for example, a **READ_ERR** with a 0 *sysiocod* value), c-treeACE Server logs the following message to *CTSTATUS.FCS* and creates a process stack dump of the c-treeACE Server process (on systems that support that ability):

```
Mon Sep 14 12:23:19 2015
 - User# 00020  READ_ERR: loc 5 file <FILENAME> offset <OFFSET> iosize <READ_SIZE_IN_BYTES> syserr
<SYSTEM_ERROR_CODE> [physical file size <FILE_SIZE_ON_DISK>]
Mon Sep 14 12:23:21 2015
 - User# 00020  Dumped stack for server process 20788, log=1, loc=0, rc=0
```

`DIAGNOSTICS READ_ERR` can be enabled and disabled at runtime by calling **ctSETCFG()** or using **ctadmn** (menu option **10, Change Server Settings**, then option **9, Change a DIAGNOSTICS option**).

DIAGNOSTICS REMAINING_THREADS

```
DIAGNOSTICS  REMAINING_THREADS
```

This keyword, when added to the server configuration file, will result in listing threads still active at server shutdown, and causing the final system checkpoint to be skipped, to the *CTSTATUS.FCS* log file. Any other internal threads will also be listed. If the final checkpoint is not skipped because of persistent client threads, then no such information is placed in *CTSTATUS.FCS*.

Example Output

```
Thu Dec 18 14:34:29 2003
 - User# 11     Remaining Thread:  ThrdID 01 ADMIN|  Atributes:00000028x
Thu Dec 18 14:34:33 2003
 - User# 11     Remaining Thread:  ThrdID 09 ADMIN|  Atributes:0000000ax
Thu Dec 18 14:34:46 2003
 - User# 11     Remaining Thread:  ThrdID 11      |  Atributes:00000000x Shutdown Thread
```

Note: Administrative threads may be expected to exist at this point in server processing. Their userid will be ADMIN. In the above example, all of these ADMIN threads were routine, and were not causing any shutdown problems. Also, the thread processing the shutdown is listed, and will be annotated as the Shutdown Thread. The attributes are for determining the identity of the administrative threads within the server.

DIAGNOSTICS REPL_READ_BUFFER

```
DIAGNOSTICS REPL_READ_BUFFER
```

Enables a check that the log data is being correctly read into the replication buffer.

DIAGNOSTICS SUBALLOCATOR_OFF

```
DIAGNOSTICS SUBALLOCATOR_OFF
```

Forces c-treeACE to allocate memory directly from the heap instead of using its internal memory suballocator.

See Also

DIAGNOSTICS MEMTRACK (page 351)

DIAGNOSTICS THREAD_DUMP

```
DIAGNOSTICS THREAD_DUMP
```

(Legacy) Enables low-level thread diagnostics.

DIAGNOSTICS TRACK_LOGON

```
DIAGNOSTICS TRACK_LOGON
```

The `DIAGNOSTICS TRACK_LOGON` option provides a net count of memory allocation requests. This count is system wide, not just the particular process logging off or logging on and requires very little overhead. See also `MEMORY_TRACK`.

Default: OFF

DIAGNOSTICS TRAN_RECOVERY

```
DIAGNOSTICS TRAN_RECOVERY
```

Causes transaction recovery log messages to be written to the file *RECOVERY.FCS* (off by default).

In the standalone model, this feature is enabled by activating `#define ctDBGRCVR` before compiling the c-tree library. In such cases, log messages are written to the file *RECOVERY.FCS*.

DIAGNOSTICS TREE_WALK

```
DIAGNOSTICS TREE_WALK
```

Enables an index tree-walk history which contains the nodes visited as well as the key comparisons made. Each time a new tree-walk is begun, the history is refreshed. If a terr() occurs, the the history is dumped into the file *LOGTREE.FCS*. This information is not maintained during high speed index loads, or during cleaning transaction information from an index.

DIAGNOSTICS UPDFLG

```
DIAGNOSTICS UPDFLG
```

Logs each change to a file update flag (*updflg*) to *CTSTATUS.FCS*.

All files are monitored when this feature is active. Each logged entry will be of the form:

```
UPDFLG: <file name> updLoc=wxyz updflg=WWx oldflg=YYx
```

The flag values are single hex bytes. *updLoc* is a four digit location code to later determine which c-tree function was actually responsible for the update.

DIAGNOSTICS WRITE_ERR_DUMP

```
DIAGNOSTICS WRITE_ERR_DUMP
```

The potential possibility of a disk write failure may go unreported with the c-treeACE Server. With a data or index file write failure, the c-treeACE Server could possibly continue operations, and assume the cache has been properly flushed to disk. While the transaction logs maintain data integrity, particular transactions may be marked as not flushed to disk, holding previous transaction logs from being discarded.

This server configuration keyword will cause the contents of the write request to be appended to the *WRITE_ERR.FCS* file. If the write is for more than 32K bytes, then only the first 32K bytes are output to the file.

To detect this situation may occur, a notification is now logged to *CTSTATUS.FCS* when and if a **WRITE_ERR** occurs. The log entry will include the file name, the offset of the write attempt, the system error code, the size of the write request and the number of bytes written. Optionally, the contents of the write request can be output to a file as well.

Note: These entries should be extremely rare!

This behavior is on by default. If this is undesirable, a new server configuration keyword will inhibit these messages:

```
CTSTATUS MASK_WRITE_ERR
```

DIAGNOSTICS WRITETHRU

```
DIAGNOSTICS  WRITETHRU
```

The c-treeACE Server writes a warning message to the *CTSTATUS.FCS* file if it detects a file that is not under transaction control and does not have *WRITETHRU* defined. This warning is to notify the developer that the file is being maintained in a vulnerable mode. Because of the overhead of writing this message to the log, and because FairCom does allow this “dangerous-cached-buffered-type” mode, the warning message is only issued once, for the first file detected. In other words, it simply tells the developer that there was “at-least-one” vulnerable file.

To help developers detect the file names for all vulnerable files, this keyword has been added. By placing this keyword in your *ctsrvr.cfg*, all file names will be listed in the *CTSTATUS.FCS* file.

Default: Disabled

11.19 Custom Server SDK Keywords

Data replication is a powerful c-treeACE feature providing near real-time data availability between one or more servers. These options configure specific attributes of this optional feature.

JOB_QUEUE_INFO (page 359)

Provides the ability to define a library (.dll or .so) followed by an entry point function that will process a job.

SERVER_SDK (page 360)

Specify options for the Server SDK. Reserved for future use.

USER_SIGNAL_DOWN (page 360)

Defines functionality to occur at Server shutdown.

USER_SIGNAL_READY (page 361)

Defines functionality to occur at Server startup.

Diagnostics

Diagnostics keywords are intended to be used to help collect data to be analyzed for troubleshooting. The use of these keywords should be avoided in normal operation as they may cause additional overhead (e.g., logging).

DIAGNOSTICS CTUSER_ERROR (page 361)

Enables errors detected while loading the *CTUSER* shared library or an error when resolving the function pointer to be logged into *CTSTATUS.FCS*.

DIAGNOSTICS CTUSER_VERBOSE (page 361)

Enables verbose information logging to *CTSTATUS.FCS* while loading the *CTUSER* shared library or when resolving the function pointer.

DIAGNOSTICS CUSTOM (page 361)

Enables custom diagnostic options to be added via the Server SDK.

JOB_QUEUE_INFO

```
JOB_QUEUE_INFO <dll:function@file>
```

Provides the ability to define a dynamic link library (.DLL or .so) followed by an entry point function within this loadable library that will process a job. The @filename.dat syntax allows you to define the c-tree job file to be processed.

Examples

```
JOB_QUEUE_INFO MYDLLNAME:myjobfunc@myjobq.dat
```

Use logic from *MYDLLNAME.DLL* (or .so), entry point function **myjobfunc()**, to process data from *myjobq.dat*

```
JOB_QUEUE_INFO RTREE@rtree.dat
```

Use logic from *RTREE.DLL* (or .so), default entry point of **ctdojob()**, to process data from *rtree.dat*

```
JOB_QUEUE_INFO MTREE@ctjobs
```

Use logic from *MTREE.DLL* (or .so), default entry point of **ctdojob()**, to process data from ctjobs - if ctjobs open fails, try *ctjobs.dat*

```
JOB_QUEUE_INFO MYJOBDLL
```

Use *MYJOBDLL.DLL* (or .so), default entry point of **ctdojob()**, to process data from default file *ctjobs.dat*

SERVER_SDK

```
SERVER_SDK  <option>
```

Current options for the Server SDK feature include:

```
HSPIPE_FROM_NET
HSPIPE_TO_MEMQ
HSPIPE_TO_DB
MEMQ_TO_DB
```

Note: This server feature is OFF by default and reserved for future use.

USER_SIGNAL_DOWN

```
USER_SIGNAL_DOWN my_parameter
```

Developers using the c-tree Server SDK may define functionality to occur at Server shutdown.

This keyword passes *my_parameter* to a user-defined function in *ctclbk.c*, which the developer should alter to suit the needs of the custom server:

```
NINT ctusersig_down(TEXT mystring)
```

This keyword can be specified multiple times with different parameters to accomplish different tasks provided the user-defined function is implemented accordingly.

Default: OFF

See Also

```
USER_SIGNAL_READY (page 361)
SIGNAL_DOWN (page 193)
SIGNAL_READY (page 193)
```

USER_SIGNAL_READY

```
USER_SIGNAL_READY my_parameter
```

Developers using the c-tree Server SDK may define functionality to occur at Server startup.

This keyword passes *my_parameter* to a user-defined function in *ctclbk.c*, which the developer should alter to suit the needs of the custom server:

```
NINT ctusersig_ready(TEXT mystring)
```

This keyword can be specified multiple times with different parameters to accomplish different tasks provided the user-defined function is implemented accordingly.

Default: OFF

See Also

```
USER_SIGNAL_DOWN (page 360)
SIGNAL_DOWN (page 193)
SIGNAL_READY (page 193)
```

DIAGNOSTICS CTUSER_ERROR

```
DIAGNOSTICS CTUSER_ERROR
```

Enables errors detected while loading the *CTUSER* shared library or an error when resolving the function pointer to be logged into *CTSTATUS.FCS*.

See Also

- DIAGNOSTICS CTUSER_VERBOSE (page 361)

DIAGNOSTICS CTUSER_VERBOSE

```
DIAGNOSTICS CTUSER_VERBOSE
```

Enables verbose information logging to *CTSTATUS.FCS* while loading the *CTUSER* shared library or when resolving the function pointer.

See Also

- DIAGNOSTICS CTUSER_ERROR (page 361)

DIAGNOSTICS CUSTOM

```
DIAGNOSTICS CUSTOM
```

Enables custom diagnostic options to be added via the Server SDK. See the modules *ctopt2.h* and *ctscfg.c*.

12. Glossary

This Appendix provides definitions for some of the terms found in this guide. Most terms are discussed in further detail throughout the rest of the guide. The definitions considered for the advanced user are depicted by listing “(advanced)” at the beginning of the definition.

Administrator
Individual typically responsible for installing, configuring, starting, stopping and maintaining the database Server and the files controlled by the c-treeACE Server.

ASCII Text File
“ASCII” is an industry code for representing characters as binary values. An ASCII Text File is a special type of file that can be copied from computer to computer and read by most word processors and editors. If this file type is unfamiliar, consult the word processor or editor documentation for additional information.

atomicity (advanced)
A term meaning an all or nothing criteria applied to data inserts, deletes and updates; with the principal goal of keeping a group of files synchronized. For example, if a record is to be updated in a series of five files as follows:

```
enable transaction
file 1 update - successful
file 2 update - successful
file 3 update - error, not updated
file 4 update - successful
file 5 update - successful
commit transaction
```

In this example, the failed update for file 3 will cause the commit transaction to fail. Atomicity indicates this failure on file 3 will prevent any updates from occurring. Without atomicity, files 1, 2, 4 and 5 would have been updated, but file 3 would not, causing the five files to be out of sync.

automatic recovery
The process of restoring files back to a pristine state after some type of catastrophe (i.e., loss of power to the computer). Automatic recovery is available only for files using full transaction processing (TRNLOG file mode). The TRNLOG file mode causes all of the changes to the particular file to be logged immediately to a special high-speed transaction log. The presence of this complete history of the changes to the data files with the TRNLOG file mode is what makes automatic recovery possible.

byte
The amount of computer memory required to store one character. To store the word “computer” takes 8 bytes. Computer memory and hard disk space are often measured in kilobytes or megabytes. 1 kilobyte = 1024 bytes (characters) = 210; 1 megabyte = 1048576 bytes (characters) = 220.

cache
A storage location where data can be more quickly accessed. For example, a disk cache will store frequently accessed data in memory to prevent reading from the slower disk drive.

c-treeACE
The c-tree file handler API that is the foundation of the c-tree database Server. c-treeACE also

serves as the client side development kit for the c-tree client/server model. c-treeACE gives a client side application the ability to communicate (connect) with the c-treeACE Server.

checkpoint
(advanced) An entry placed in c-treeACE Server logs identifying a starting point during the Server's automatic disaster recovery.

client
The second half of the client/server model. The client process is typically a third party application performing a specific task. For example, in a client/server based accounting system, the program prompting the user for input and displaying results would be considered a client.

ctadmn
c-treeACE Server utility program allowing the Server Administrator to grant access to the Server through user identification names. Controls high-level access to the Server and should be executed by the Server Administrator only.

ctdump
c-treeACE Server utility program for creating file backups. This unique utility allows the backup of user data to be performed without restricting user access in any way.

ctfdmp
c-treeACE Server utility program for rolling forward from a specific point in the transaction log history. The c-treeACE Server has the ability to allow users to reset files to specific points in time. This utility allows older backups of user data to be brought up to date.

ctldmp (advanced)
c-treeACE Server utility program for displaying detailed information from the Server transaction logs.

ctrdmp
c-treeACE Server utility that restores the backups made using ctdump. This utility also performs data rollbacks.

ctpass
c-treeACE Server utility to allow users to change their password.

ctstop
c-treeACE Server utility for stopping the c-treeACE Server.

data file
A collection of similar pieces of information stored in one location.

deadlock (advanced)
A situation in which two users are prevented from obtaining access to the same record. An example of this situation is if user A locks record 1 for update. At the same time, user B locks record 2 for update. User A now needs to lock record 2 and user B needs to lock record 1. If user A and B both need locks on their target records before processing can continue, a deadlock occurs. The c-treeACE Server automatically handles this situation by returning error messages to the users involved.

directory
A location where files are stored on disk. A directory can be thought of as a hanging folder in a file cabinet. Each file folder within the hanging folder can be thought of as a separate file, or collection of information.

dynamic dump
A method for backing up specified files without restricting user access to the files.

encryption
Disguising information making it difficult for a casual user to inspect the actual contents.

F_TCPIP
The COMM_PROTOCOL keyword for specifying the TCP/IP transport protocol. A DLL of this name was included prior to c-treeACE V11.2 and c-treeRTG V2 (beginning with those releases, the logic is embedded into the server and the DLL is not shipped).

F_TCPIPV6
The COMM_PROTOCOL keyword for specifying the TCP/IP transport protocol using an IPv6 socket. A DLL of this name was included prior to c-treeACE V11.2 and c-treeRTG V2 (beginning with those releases, the logic is embedded into the server and the DLL is not shipped).

FETCPIP
The COMM_PROTOCOL keyword for specifying the TCP/IP transport protocol with encryption. A DLL of this name was included prior to c-treeACE V11.2 and c-treeRTG V2 (beginning with those releases, the logic is embedded into the server and the DLL is not shipped).

FSHAREMM
The COMM_PROTOCOL keyword for specifying the shared memory transport protocol. A DLL of this name was included prior to c-treeACE V11.2 and c-treeRTG V2 (beginning with those releases, the logic is embedded into the server and the DLL is not shipped).

file
A collection of related information, referred to as records. See the definitions for directory and record for further information.

folder
The Apple Macintosh term for a location of files on disk. Similar to directory defined above.

hash bins (advanced)
A mathematical algorithm for storing data in memory so it can be quickly located and retrieved. The c-treeACE Servers all use sophisticated hashing routines for data and index caches.

index file
A special type of file used by c-treeACE and the c-treeACE Server for quickly locating information (records) within a data file.

log (ctstatus, transaction)
A special purpose file containing important information about a specific process. For example, the c-tree *CTSTATUS* log will contain status information about the Server. Storing items such as when the Server was last started and stopped and what files have been backed up using the dynamic dump facility, etc.

logging
The process of keeping a permanent record of the changes made during a transaction.

message queues
A communication protocol typically used on Unix based operating systems. This communication protocol allows a client process to talk to a c-treeACE Server process.

mirroring
A mechanism for duplicating important files on different hard drive volumes, partitions or physical

devices. If the primary storage location is lost due to some form of catastrophe (i.e., hard disk crash) the mirroring logic can automatically detect the lost connection and switch to the secondary or “mirrored” storage area without any user intervention.

NetBIOS (Network Basic Input/Output System)
A communication protocol that was typically used on legacy Microsoft based operating systems. This communication protocol has been deprecated.

page (advanced)
A unit of measure for electronic data. At the lowest level operating system read and write, data is manipulated in page sizes. A common page size for Windows and Unix is 4096 bytes.

PREIMG (advanced)
A transaction processing file mode supporting atomicity, but not automatic recovery.

process
For purposes of this Guide, a process is equivalent to a user or connection. One process equals one user.

record
A piece of information stored within a file. Expanding on the file cabinet example used in the directory definition, each piece of paper found within a file folder can be thought of as a record. A record is a unique piece of information similar to other pieces of information (papers) within the same file folder.

roll back
A process made possible with transaction processing allowing file transactions to be reset back to a specific point in time. For example, if a data entry operator enters two hours worth of transactions with incorrect information, these transactions can be removed or rolled back.

roll forward
Similar to roll back, except moving forward in time. The roll forward process is typically for applying transactions to “out dated” (old) files. To do this transaction processing logs containing the desired transactions must be available.

semaphore (advanced)
An operating system level counter used for controlling access to a limited pool of shared resources, such as shared memory.

server
The term server can refer to many different items. For example a “file server” is a specific piece of software for sharing files and hardware devices among users. A “hardware server” is a special computer on which the file server operates. The c-treeACE Server is a “database server”, special software efficiently managing multiple users accessing common data.

shared memory
A communication protocol typically used on Unix and Windows based operating systems. This communication protocol allows a client process to talk to a c-treeACE Server process.

superfile
A physical file containing any number of logical data and index files.

tar
The tar command is used for creating and managing file backups. c-treeACE Unix Servers are

shipped in tar format. The tar command is used to copy the files from the distribution floppies to the hard drive.

TCP/IP
Transport Control Protocol/Interface Program. A communication protocol available on most operating systems. This communication protocol allows a client process to talk to a Server process.

transaction
A specific operation on a file (i.e., adding a record, deleting a record, updating a record - are all examples of a transaction).

transaction processing
A mechanism by which several data integrity issues are handled. Two of the most important issues are atomicity and automatic recovery.

user
A client process (application) connected to a c-treeACE Server. Each process connected to a c-treeACE Server is counted as a user.

working directory
The directory pointed to by the `LOCAL_DIRECTORY` (page 170) keyword in *ctsrvr.cfg*, or (if not specified) the directory where the c-treeACE Server process resides.

13. Index

D

E

F

T

U

V

W

X