



FairCom®

c-tree **Plus**®
V9

c-treeACE V9.0 Release
Notes

c-treeACE V9.0 Release Notes



CONTENTS

c-treeACE V9.0 Release Notes	1
c-treeACE SQL	3
2.1 Critical Issues.....	3
c-treeSQL Server Buffer Overruns Corrected.....	3
Checkpoint Error 7054 Corrected with the c-treeSQL Server	3
c-treeSQL Server Panic Corrected	3
Resolved c-treeSQL Server Panic when using Parameters in a Predicate Clause	4
c-treeSQL Client Connections no Longer Hang when the Server Connection Limit is Reached	4
c-treeSQL Server Daemon:accept error Corrected on Solaris and SCO Operating Systems.....	4
Improved Argument Handling With c-treeSQL Stored Procedures	4
c-treeSQL Server Commits During Cost Calculations	4
2.2 Serious Issues	6
Improved c-treeSQL Server Query Stability	6
First Row of LONG VARCHAR Fields Correctly Updated	6
Automatic Recovery Error L64 Corrected.....	6
LOCAL_DIRECTORY Server Keyword Behavior Corrected	6
Proper Handling of SQL SYNONYMs.....	6
RECBYT Index for First Index of Table	7
Qualified User Defined Functions Fixed	7
Proper Handling of Maximum Number of Connected Clients.....	7
Microsoft Access Credentials with c-treeSQL ODBC	7
Correct Queries with Scalar Function and Parameters in a WHERE Clause.....	7
Correct Query Results with Self Join and Column Aliases	8
Corrected LEFT OUTER JOIN Optimization when Used as a Predicate Clause	8
Correct Handling of non-US ASCII Characters in Stored Procedure Java Strings	8
Proper Joins with c-treeSQL LONGVAR Fields.....	8
Improved TCP/IP Handling of accept() Call Failures	8
c-treeSQL BIGINT Byte Order Change on HIGH_LOW Architectures	9
Eliminated DLOK_ERR (42) During c-treeSQL INSERT	9
2.3 Other Issues.....	10
LONG VARCHAR Fields Properly Updated	10
CT_BOOL Correctly Maps to SQL_BIT with SQL Table Import.....	10
UPPER() Function Syntax Corrected	10
c-treeSQL Log File no Longer Prevents Client Connection.....	10
Correct Linking of Tables in MS Access	10
Dropping a Constraint Executed by a User Other than the Constraint Creator.....	10
Correct Query Results Returned When Using an OR Conjunction	11
c-treeSQL ODBC SQLStatistics() Returns Correct Information	11
Correct Number of Affected Rows Returned with SQLGetDiagField()	11
Proper Return of LVARCHAR Data With SQLExecDirect().....	12
Removed Required DSN Connection Strings for the Direct Link ODBC Driver	12
Correct Array Handling with Prepared Statements in JDBC	12
More than 2000 Bytes Allowed with JDBC LVARCHAR Retrievals	12
Proper Batch Resizing with the JDBC Driver.....	12
Proper Batch Additions with VARBINARY Fields	12
Correct JDBC Exception now Thrown when User Limit is Exceeded.....	13
c-treeSQL Date Field Range.....	13
Correct Join Handling in Queries	13
Proper Cursor Close with Stored Procedure Resultsets	13
c-treeSQL Server System Signal Handling.....	13

Retrieval of LVARCHAR Fields Using the c-treeSQL ODBC Driver and ADO .NET	13
Improved JDBC and ODBC Driver Handling of Empty Resultsets with Long Fields.....	14
Proper Semicolon Syntax with ISQL.....	14
Eliminated Overflow/Underflow Errors Involving Joins	14
Corrected SELECT COUNT(*) from a Variable Length Table Without an Index.....	14
Corrected JDBC -21043 Error.....	14
Proper LVC Update with the c-treeSQL JDBC Driver in Autocommit Mode	15
Improved c-treeSQL JDBC Performance when Resizing the SQLDA Object	15
Physical File Name Changed with c-treeSQL RENAME TABLE Statement	15
Increased Number of Network Backlog Users Allowed	15
Deferred Key Add Operations For Update Of Transaction-Controlled Unique Index.....	15

c-treeACE Server	17
3.1 Critical Issues.....	17
c-tree Server Hang Corrected.....	17
Improved Shared Library Support on Linux	17
Fatal Log Write no Longer Hangs Server at Shutdown	17
CTSTATUS_SIZE and DIAGNOSTICS LOWL_FILE_IO Configuration Options no Longer Hang Server	18
MEMORY_MONITOR Option no Longer Crashes Server on Windows 2000.....	18
Server no Longer Crashes when Passed a datno from BlockingISAMRead	18
Maximum Connected Client Limit no Longer Terminates the c-tree Server.....	18
"Abort node error" When Closing PREIMG files	19
3.2 Serious Issues	20
WRITE_ERR Notification	20
Correct Path References with the Novell NLM	20
Record Conversion Buffer Overflow Fixed	20
Improved Dynamic Dump of HUGE files for c-tree Servers on Linux.....	21
Dynamic Dump Includes Matching Filenames with Wildcards and LOCAL_DIRECTORY Configuration Keyword	21
Proper Server Shutdown with the ADMIN_ENCRYPT Configuration Keyword	21
Compacting HUGE Files with the c-tree Server Properly Returns all Records	21
Corrected Connection Memory Leak	21
3.3 Other Issues.....	23
c-tree Server Timeout Feature on Shutdown.....	23
File System Cache Now Flushed Before Physical Close	23
Dynamic Dump Detects Incorrect Header Values for non-HUGE Files.....	23
Update of Internal Unique File ID Corrected.....	23
Correct Key Transformations with the Unicode Enabled Server	24
VIRTUAL Open of Segmented File Corrected.....	24
"Improper Shutdown Detected" Dialog Suppressed	24
Proper Handling of Log Template Rename Failures	24
SNAPSHOT Negative Statistics Output Corrected.....	24
Correct Zero Timeout Handling with ctSysQueueRead.....	25
Correct Return Values of System Queue Functions.....	25
Update when a SCHSEG key Segment is Defined on a Unicode Field	25
Automatic Switch to O_SYNC in Case of Direct I/O Error	25
Correct SIGNAL_READY Syntax with Long Paths.....	26
Invalid Space Management Index Root After Recovery.....	26
TMPNAME_PATH for Rebuild Sort Work Files now Operates as Expected with the c-tree Server.....	26
Correct GTEKEY Retrieval with 64-bit Memory File Key Values.....	26
Read-Only Dynamic Dump Script Files	26
Correct Index Recovery After a Checkpoint Splits a Transaction Commit	26
SEQ8_ERR Added to System Message Management File.....	27

Additional Errors Added to Server Shutdown Exclusions	27
c-tree Server Defunct Process Cleanup with SIGNAL_READY Handling on Unix Systems	27
<hr/>	
c-treeACE for .NET	29
4.1 New Dispose() Method for c-tree Plus for .NET	29
4.2 Resolved Conflict Between ctNUMBER and CTNumber class	29
4.3 Correct Return Types for GetFieldAsFloat()	29
<hr/>	
c-treeDB	31
5.1 Critical Issues.....	31
Memory Leak in ctdbFreeRecord.....	31
Disappearing Records with ALTER_TABLE Resolved.....	31
5.2 Serious Issues	32
Dictionary Operations Now Properly Keeps All Locks.....	32
ctdbBegin no Longer Fails When CTKEEP_MODE is Active.....	32
Incorrect Index File Extension with Alter Table Corrected.....	32
Proper AtterTable() of non-HUGE Files	32
c-treeDB Path Prefix Now Properly Constructed.....	33
ctdbWriteRecord no Longer Fails when Unicode Segment is Defined.....	33
c-treeDB Header and IFIL Modes now Match After Alter Table	33
Table Open with CTOPEN_READONLY Disallows Record Add and Update.....	33
5.3 Other Issues.....	34
c-treeDB now Removes .FCZ Files After Alter Table Operation	34
c-treeDB Index Name can now be Changed	34
c-treeDB Now Returns Specific Errors for Duplicated Index Names.....	34
Import of CT_DATE Field with Index Segment Mode of SGNSEG	34
ctdbGetRecordBuffer() Buffer Return Corrected	35
File Extension Detection Logic Now Working Correctly for	
Empty Extensions when Importing Files with ctsqlimp	35
ctdbGetFieldNumber Does not Properly set the Error.....	35
Owner Name Case Handling Corrected in c-treeDB	35
Existing Databases Properly Checked when calling ctdbAddDatabase.....	35
Correct Alter Table With a CT_FSTRING Field Length of 256.....	35
CT_CURRENCY now Valid for SNGSEG Segment Mode.....	36
Proper Connect to a Database Created Inside a Transaction.....	36
c-treeDB now Correctly Converts String to CTMONEY.....	36
Corrected CTNumber Declarations.....	36
Corrected Handling of Empty Variable Length Fields with c-treeDB	36
Properly Built c-treeDB Target Keys with Conditional Indexes.....	36
CTDBRET_NOSEGMENT Returned on Alter Table and Create Table	37
ctdbFindRecord() with CTFIND_EQ mode and Indexes Allows Duplicates	37
<hr/>	
c-treeACE.....	39
6.1 Serious Issues	39
FPUTFGET Locking Issues	39
Proper Reference of Unused File Numbers with CLSFIL.....	41
ISRL_ERR (554) with SCHSRL During Rebuild Corrected	41
MACOSX - XCODE Syntax Error	41
terr(8989) Under FPUTFGET Corrected	42
Calling GetSymbolicNames() Now Returns Correct File Name	42
Complete Update of File Sync List from Notification	42
EQLKEY Now Checks for Invalid keyno	42
Unused Debugging Code Removed in Hugefile Support	42
ctLOCLIB with ctCLIENT Redirection Solved	42
LOCLIB Memory Overwrite Fixed.....	42

ctSysQueueRead Hang on Empty Queue Corrected	43
Access Violation on c-tree Superfile Member Create or Open.....	43
Extended Create Block Function Corrected	43
Correct File Open with a Unicode Key Segment on an Index Member.....	43
Correct Fileword Corruption in OPNRFILX.....	43
6.2 Other Issues.....	44
Current ISAM Positions after Batch Insertions	44
Records on LOW_HIGH Platforms	44
Record Offsets of Zero for Physical Order Batch Reads Supported	44
Support System Queue Functions in LOCLIB mode	44
Correct Return Values of System Queue Functions.....	44
VDLK_ERR During RETVREC Call in Single User Mode	44
Proper Deny Open of a Mirrored Pair of Segmented Files.....	45
Transaction Control of Conditional Index Expressions Corrected	45
FreeRange (FRERNG) Header File Fix.....	45
Assignment of updateIFIL Now Preserves Previous IFIL Setting.....	45
Old fileIDs Properly Properly Removed During Rebuild	46
Corrected Serial Segment Test.....	46
BLKIREC Properly Finds all Matching Records.....	46
LTEREC Rproperly Returns Record with Key Value Equal to Target	46
vtclose no Longer Fails in FPUTFGET Mode.....	47
c-tree Status Log Write Failure Causes Process to Exit.....	47
Support Range Functions when Huge File Support is Disabled.....	47
Compiling the Standalone Model Without c-tree Superfile Support	47
Correct Open of File with Unicode Key Segment	47
Transaction Commit no Longer Fails with SEEK_ERR (35).....	48
Improved Sensitivity of ESTKEY Routine	48
ISAM Context Data Properly Updated During Index Create or Delete	48
Correct 8-byte Serial Number Usage.....	48
REDIREC() no Longer Fails in Multi-User Multi-Threaded Mode.....	48
Rebuild Now Correctly Purges Records with Duplicate Keys when c-tree is Initialized with USERPRF_PTHTMP.....	49
Stable Index Sizes after Automatic Recovery.....	49
ctCAMO support for c-tree Client Compiles.....	49
Rebuild Callback with LOCLIB Mode.....	49
Correct NTIM_ERR Returned by SYSMON() when a Time-out Occurs	49
Correct Byte Ordering of nul Padded CT_STRINGs with the c-tree Plus ODBC Driver in Heterogeneous Environments.....	49
Corrected Lock Dump Output	50
Properly Initialized CMPIFIL() Callback Variables	50
Improved Handling of Header Values Prevents KDUP_ERR with SRLSEG Keys.....	50
c-treeACE Utilities.....	53
7.1 Successful Rebuild of Segmented Files With no File ID	53
7.2 Correct Restore of a Segmented Dynamic Dump File	53
7.3 Avoid Unnecessary XCRE_ERR Error During Rebuild with Missing Indices	53
7.4 c-tree Restore Utility Opens all ctTRNLOG files Restored from Dump	53
7.5 IFIL-Based Rebuild Utility Correct Functions with Both -purge and -updifil.....	54
7.6 Flush Standard Output Before ctclntrn and cthghtrn Utilities Exit.....	54
7.7 dbload Memory Overwrite Corrected.....	54
7.8 c-treeSQL dbdump Crash Corrected	54
7.9 Correct Monitoring of a Large Number of Clients with the c-tree Statistics Utility.....	54
7.10 ctstat Utility Returns Exit Code 101	54
7.11 sa_admin now adds New User Accounts to Specified Group	55
7.12 ctstop Utility now Exits with a Non-Zero Return Code.....	55

7.13 Dump Index Utility Support for Huge Files..... 55

7.14 c-tree Configuration Settings Utility no Longer Requires Write Permissions 55

7.15 Correct Mirrored File Handling with the Superfile Prepass Utility..... 55

7.16 Correct Usage Output for IFIL-Based Rebuild Utility..... 56

Index..... 57

Copyright © 1992-2008 FairCom Corporation All rights reserved. No part of this publication may be stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of FairCom Corporation. Printed in the United States of America.

Information in this document is subject to change without notice.

Trademarks

c-tree, c-tree Plus, r-tree, the circular disk logo, and FairCom are registered trademarks of the FairCom Corporation. c-treeACE SQL, c-treeACE SQL ODBC, c-treeACE SQL ODBC SDK, c-treeVCL/CLX, c-tree ODBC Driver, c-tree Crystal Reports Driver, c-treeDBX, and c-treePHP are trademarks of FairCom Corporation. The following are third-party trademarks: AMD and AMD Opteron are trademarks of Advanced Micro Devices, Inc. Macintosh, Mac OS, and Xcode are trademarks of Apple Inc., registered in the U.S. and other countries. Borland, the Borland Logo, Delphi, C#Builder, C++Builder, Kylix, and CLX are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. Business Objects, the Business Objects logo, Crystal Reports, and Crystal Enterprise are trademarks or registered trademarks of Business Objects SA or its affiliated companies in the United States and other countries. DBstore is a trademark of Dharma Systems, Inc. HP and HP-UX are registered trademarks of the Hewlett-Packard Company. AIX, IBM, OS/2, OS/2 WARP, and POWER5 are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. Intel, Itanium, Pentium and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. LynuxWorks, LynxOS and BlueCat are registered trademarks of LynuxWorks, Inc. Microsoft, the .NET logo, MS-DOS, Visual Studio, Windows, Windows Mobile, Windows server and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Novell and NetWare are registered trademarks of Novell, Inc., in the United States and other countries. QNX and Neutrino are registered trademarks of QNX Software Systems Ltd. in certain jurisdictions. Red Hat and the Shadow Man logo are registered trademarks of Red Hat, Inc. in the United States and other countries, used with permission. SCO and SCO Open Server, are trademarks or registered trademarks of The SCO Group, Inc. in the U.S.A. and other countries. SGI and IRIX are registered trademarks of Silicon Graphics, Inc., in the United States and/or other countries worldwide. Sun, Sun Microsystems, the Sun Logo, Solaris, SunOS, JDBC, Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. UNIX and UnixWare are registered trademarks of The Open Group in the United States and other countries. Linux is a trademark of Linus Torvalds in the United States, other countries, or both. All other trademarks, trade names, company names, product names, and registered trademarks are the property of their respective holders.

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

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

Portions © 1987-2008 Dharma Systems, Inc. All rights reserved. This software or web site utilizes or contains material that is © 1994-2007 DUNDAS DATA VISUALIZATION, INC. and its licensors, all rights reserved.

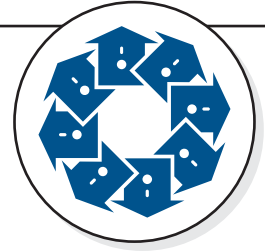
6/26/2008

FAIRCOM TYPOGRAPHICAL CONVENTIONS

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

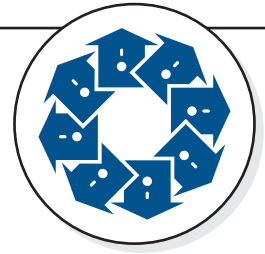
The following formatted items identify special information.

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.
<i>FairCom Terminology</i>	FairCom technology term.
FunctionName()	c-tree Function name.
<i>Parameter</i>	c-tree Function Parameter.
Code Examples	Code example or Command line usage.
utility	c-tree executable or utility.
<i>filename</i>	c-tree file or path name.
CONFIGURATION KEYWORDS	c-treeACE Configuration Keyword.
BIG_ERR	c-tree Error Code.



c-treeACE V9.0 Release Notes

Numerous fixes have been identified and corrected following the V8.14 release of c-tree Plus and c-tree Server. In particular, significant work has been done to achieve even greater server stability. Several critical memory buffer overruns were identified through extensive profiling and quality assurance testing. Other issues identified through technical support contact with our customers have also been addressed and corrected in this latest version. FairCom's outstanding support is always there to help when you encounter a problem.



c-treeACE SQL

2.1 Critical Issues



c-treeSQL Server Buffer Overruns Corrected

Extensive QA testing of the c-treeSQL Server revealed several instances where a buffer overrun could occur. In addition, several uninitialized variables were also found which could cause invalid pointer references leading to a server crash. Numerous changes were enacted greatly enhancing the stability and reliability of the c-treeSQL Server.

Checkpoint Error 7054 Corrected with the c-treeSQL Server

A c-treeSQL client connecting at the same time a checkpoint is occurring changed user information the checkpoint thread was processing. The checkpoint thread detected this change and terminated the server process with an internal error (terr 7054).

Logon logic has been modified to acquire a proper mutex before changing user information, thus preventing the checkpoint thread from detecting this timing-sensitive error.

c-treeSQL Server Panic Corrected

A Panic condition occurs when the c-treeSQL server detects an internal condition that cannot be resolved and would lead to unpredictable results. The server attempts a controlled shutdown to preserve as much information as possible. The server Panic produced the following information in *CTSTATUS.FCS*:

```
Tue Sep 13 16:50:43 2005
- User# 00017: ERROR - - TPEUTIL error 201 errno 0
Tue Sep 13 16:50:43 2005
- User# 00017: PANIC - AUTH auth_dba_seluser ssm_open failed PID 3956
Tue Sep 13 16:50:43 2005
- User# 00017 SQL PANIC, attempting shutdown
```

The supply of cursors available for server use had become exhausted when a cursor had been required for internal operations. This internal request for a cursor failed with error **20125** and the Panic ensued. This has been resolved by maintaining an internal pool of server cursors.

Resolved c-treeSQL Server Panic when using Parameters in a Predicate Clause

A c-treeSQL Panic was observed in production when given a query with parameters in the predicate clause such as:

```
'f1 >= scalar_func(?) AND f1 <= scalar_func(?)'
```

The c-treeSQL optimizer encountered a situation where the parsing tree became unstable. The optimization engine was modified to properly handle this situation.

c-treeSQL Client Connections no Longer Hang when the Server Connection Limit is Reached

An internal deadlock was found when clients connected to a c-treeSQL Server and the server's connection limit had already been reached. This behavior was only observed on the Solaris operating system. Additional checks on the acquired mutex are now done to ensure this situation does not occur. An additional check was added to output the following message to *CTSTATUS.FCS* when this situation occurs in other situations:

```
os_acquire_mutex_lock detected recursive call
```

The c-treeSQL Server will then force a process stack dump to be output. Only the first ten failed mutex request calls will result in the stack dump.

c-treeSQL Server Daemon:accept error Corrected on Solaris and SCO Operating Systems

The following message was observed on standard output from the c-treeSQL Server:

```
Daemon:accept failed: Software caused connection abort
```

From this point on, c-treeSQL clients failed to connect to the c-treeSQL Server, reporting the following error:

```
-20212 "Error in Network Daemon".
```

This behavior was observed on Solaris and SCO operating systems. When a SQL client terminated its connection to the c-treeSQL Server after the server has queued its connection request to be accepted BUT before the c-treeSQL Server has accepted the connection, the socket **accept()** call fails with error *ECONNABORTED*, and the c-treeSQL Server's SQL communication listener thread exited its listen loop. To resolve this, we now treat error *ECONNABORTED* as a recoverable error, and in this case, retry the **accept()** call.

Improved Argument Handling With c-treeSQL Stored Procedures

A c-treeSQL "Panic" could be encountered when running the same stored procedure multiple times. The panic resulted from a memory overwrite that could occur when executing a stored procedure and the sum of the number of arguments and resultset columns exceeded a maximum value of 50. A check is now made to ensure that neither the procedure arguments nor the result set columns exceed their individual maximum values of 50 each resulting in a new total combined number of 100. An appropriate error is now returned when this condition fails.

c-treeSQL Server Commits During Cost Calculations

A Dr Watson log of a c-treeSQL Server crash revealed a fault where memory was freed twice. It was possible where the same memory was assigned to two different pointers. This situation was extremely

unusual as only during the commit phase of a transaction while the c-treeSQL engine was checking index costs and the commit changed an index resulting in a cost change could this occur. The c-treeSQL Server now correctly frees only the allocated memory.

2.2 Serious Issues



Improved c-treeSQL Server Query Stability

Several types of specific c-treeSQL query statements could result in one of the following:

1. error **-20008**, "Inconsistent types":
2. error **-20232**, "Invalid number string":
3. The c-treeSQL Server terminated with an unhandled exception:

The query optimization failed for these queries due to an error in the c-treeSQL engine's optimization logic. Correcting the c-treeSQL engine optimization now allows these queries to execute successfully.

First Row of LONG VARCHAR Fields Correctly Updated

A customer reported that only the first row was updated upon a c-treeSQL UPDATE statement with a LONG VARCHAR (LV) field. Internally, LV fields are handled somewhat differently than other fields as LV fields can be enormous. Specifically, a handle is used to refer to the field rather than the actual contents passed around. This handle was initially bound to a field, and then improperly reused, resulting in only the first row receiving the update, regardless of which record was in process. An inappropriate check was removed to correct this behavior.

Automatic Recovery Error L64 Corrected

A reproducible situation was reported where the c-treeSQL Server produced a matching file ID resulting in a L64 automatic recovery failure. Creating a new c-treeSQL database involves copying the template c-tree Superfile to a new directory with a new c-tree Superfile name using a system copy. The file IDs of the c-tree Superfile host and its members were then changed, however, some or all c-tree Superfile members could be skipped in this operation. An uninitialized variable was identified, and corrected.

LOCAL_DIRECTORY Server Keyword Behavior Corrected

A **DCRAT_ERR** (17) error was reported at server startup during the template database creation when using the c-tree Server **LOCAL_DIRECTORY** configuration keyword and a relative path. Using an absolute path for this keyword worked properly. When **LOCAL_DIRECTORY** is specified, the c-tree SQL Server previously changed the working directory to the **LOCAL_DIRECTORY**. The server had improperly pre-pended directory information to the given path, causing an invalid path to be formed. The **LOCAL_DIRECTORY** keyword now functions as expected.

Proper Handling of SQL SYNONYMS

When using the c-treeSQL Server, it was found the handling of synonyms was inconsistent depending on the user. When created by one user, the synonyms were not consistency available to another user,

even though permissions were properly granted. The tables were not being resolved properly. The use of synonyms now functions as expected.

RECBYT Index for First Index of Table

When using the c-treeSQL Import Utility, **ctsqlimp**, there can be a problem with the *RECBYT* index if it is not the first index. When importing a table with a *RECBYT* index named "\$RECBYT\$" and the index was not the first index of table, the c-treeSQL Import Utility failed with error **CTDBRET_INTERNAL** (4025). The c-treeSQL import utility had a dependency on the *RECBYT* index being the first index of a table and this dependency has been removed from the utility.

Qualified User Defined Functions Fixed

A qualified User Defined Function (UDF) is one where the function name follows the owner name like owner.function. A syntax error running a simple query when using a qualified UDF could be generated when a function returns a value as a parameter.

For example, the following c-treeSQL statement would not work:

```
INSERT INTO someuser.SOME_USER_TABLE1 VALUES ('Data1', someuser.TO_NATIVE_DATE(cast('1115755417'
as INT)));
```

The c-treeSQL parser was not properly handling this construct and this has been corrected.

Proper Handling of Maximum Number of Connected Clients

If a c-treeSQL client attempts to connect to the c-treeSQL Server and the maximum number of clients are already connected to the server, the c-treeSQL client would hang. When a c-treeSQL client connects, the c-treeSQL Server attempts to create a thread. When the maximum number of allowed connected clients are attached, this thread was not created and the server failed to return an error to the client. As a result, the communications socket was held open and the client would wait indefinitely, resulting in the hang.

An error code (**-17084**, users exceeded) is now returned such that the client realizes the connection attempt has failed and the client can close the connection. This error message is included in the c-treeSQL client's error list so the client can display an error message when this error occurs.

Microsoft Access Credentials with c-treeSQL ODBC

When connecting to the c-treeSQL ODBC Driver using Microsoft Access, if the driver's data source did not specify a user name or password, the driver prompted the user for the user name and password, but the connection attempt failed even when the correct user name and password were specified. Internally, the c-treeSQL ODBC driver improperly changed the data source name which resulted in the failed connection. This behaviour has been corrected.

Correct Queries with Scalar Function and Parameters in a WHERE Clause

The following statement would fail when executed against a c-treeSQL Server:

```
SELECT TOP 50 name, id FROM customers B WHERE UPPER(B.name) >= ? ORDER BY B.name
```

A case of query statements failed when the *WHERE* clause contained a scalar function and a parameter(s). The c-treeSQL Server did not properly handle parameters with scalar functions. This limitation has been removed, and the above query will now execute correctly.

Correct Query Results with Self Join and Column Aliases

A `SELECT` statement containing a self table join with column aliases and a column with the same name of an alias assigned to a different column, returned the wrong result. This construct now properly returns the correct result.

Corrected LEFT OUTER JOIN Optimization when Used as a Predicate Clause

A particular query was found which did not return a correct result set when a `LEFT OUTER JOIN` was specified as the predicate:

```
SELECT p.*, ip.STUDENT_ID, ip.IS_NULL, ip.STRING_VAL, ip.NUMBER_VAL, ip.TIME_VAL, v.VAL_NAME,
v.SEQ_NUM AS VAL_SEQ_NUM, cp.IS_REQUIRED
FROM CT_CLASS_PROP cp
INNER JOIN CT_PROP p ON p.PROP_ID = cp.PROP_ID
LEFT OUTER JOIN CT_INSTANCE_PROP ip ON p.PROP_ID = ip.PROP_ID
    AND STUDENT_ID = '12345'
LEFT OUTER JOIN CT_PROP_VAL v ON v.prop_val_id = ip.STRING_VAL
    AND p.PROP_TYPE = 5
WHERE CLASS_ID = '67890'
ORDER BY CLASS_ID, cp.SEQ_NUM;
```

The query returned 0 rows while the correct result should have been 1 row. The query optimizer had moved a predicate clause from the last `LEFT OUTER JOIN` to the `INNER JOIN`, inappropriate when the predicates are part of a `LEFT OUTER JOIN` condition. Logic to check this `LEFT OUTER JOIN` predicate condition has been added resulting in a proper query return.

Correct Handling of non-US ASCII Characters in Stored Procedure Java Strings

When returning a string from a stored procedure, if the string contained non-US ASCII characters, the returned string contained garbage. An internal conversion is now done with strings within the Java stored procedure code to ensure all characters are properly converted.

Proper Joins with c-treeSQL LONGVAR Fields

It was found that a c-treeSQL query joining two tables and selecting a `LONGVAR` field could return the wrong result. When performing a join, c-treeSQL may not fetch the `LONGVAR` field until the end of the join process when the field handle has already been closed. An error that should have been returned in this case was not properly passed to the client, thus resulting in an incorrect return value.

To correct the problem of the closed field handle, c-treeSQL now stores the table `uid` allowing a later fetch to properly re-open the table and use the appropriate field handle. Should this field handle unexpectedly become invalid, the error condition is also now properly passed to the client application.

Improved TCP/IP Handling of accept() Call Failures

A particular environment found c-treeSQL clients failed to connect to the c-treeSQL Server. The system `netstat` utility did not show a c-treeSQL Server port in use. The c-treeSQL Server's standard output displayed a message such as:

```
Daemon:accept failed: Protocol error
```

When the system TCP/IP `accept()` call fails, and if the error code is `EINTR`, `EWOULDBLOCK`, `ECONNABORTED`, or `ETIMEDOUT`, the c-treeSQL Server retried the `accept()` call. For other error codes, the server closed the socket, and this prevented c-treeSQL clients from connecting.

The c-treeSQL Server now always retries the `accept()` call when it fails, unless the server is shutting down. If more than 100 consecutive calls to `accept()` fail, the server logs the following error message to standard output as well as `CTSTATUS.FCS`:

```
SQL connection accept failed (<number_of_attempts> attempts), socket=<socket_descriptor>: <error code>
```

The server will then pause for one second before attempting the next `accept()` call.

c-treeSQL BIGINT Byte Order Change on HIGH_LOW Architectures

An imported c-treeSQL data table produced unexpected queries on a particular unsigned field on HIGH_LOW architectures (Solaris Sparc in this case.)

The c-treeSQL import utility promoted a `CT_UINT4` field to a BIGINT which is treated as a native eight byte integer. The c-treeSQL Server, however, incorrectly byte swapped BIGINT values stored on disk with HIGH_LOW systems. The c-treeSQL Server assumed the imported data was already byte swapped and the data value was again byte swapped when read, producing the incorrect results.

When data was only added and read by the c-treeSQL Server there were no problems noted, as the c-treeSQL Server swapped the bytes when the data was read back.

Note: This is not an issue with c-tree Plus data files, or c-tree Plus ISAM only applications. Only c-treeSQL storage of the BIGINT field type on HIGH_LOW architectures are affected.

There are two important issues that result from this incorrect data storage.

1. Indexes built over BIGINT fields created by c-treeSQL on HIGH_LOW systems will be incorrect, as the byte order is improperly reversed.
2. Imported existing c-tree Plus tables which have correct eight byte integers, `CT_UINT4` values, stored on disk will return incorrect data (incorrectly byte swapped) when queried by c-treeSQL.

To resolve this, BIGINT byte swapping logic has been modified to ensure the byte order of the underlying architecture is properly considered.

Compatibility Issues

This change introduces a compatibility issue with previous existing c-treeSQL tables where a BIGINT field on HIGH_LOW systems is exclusively added and maintained in c-treeSQL.

Only applications which store BIGINT values created by c-treeSQL on HIGH_LOW platforms are affected.

A utility is available from FairCom to correct tables which exhibit this incorrect storage. Contact your nearest FairCom office should you require such assistance.

Eliminated DLOK_ERR (42) During c-treeSQL INSERT

An unexpected `DLOK_ERR` (17042) occurred during a c-treeSQL INSERT operation.

The coalesce space logic, for variable length files, can walk onto and lock space just allocated by the file extension functions for a new record. The new record logic's attempt to lock the space then fails. Locking logic in the file extension functions has been modified to avoid this undesired locking behavior for new records.

2.3 Other Issues

LONG VARCHAR Fields Properly Updated

A customer reported an issue when updating LONG VARCHAR fields. Consider the following where there is a variable length text field that has the following data in it:

Thisisareallylongwordstoredinavariablengthtextfield

An update is done on this field through a JDBC prepared statement with the following update:

```
update tablename set text=? where num=5"
```

If the value "Muchsmallerword" is submitted and field is re-queried, it could look as the following:

Muchsmallerwordordstoredinavariablengthtextfield

The data was overlaid rather than replaced. This condition is now properly handled with the c-tree JDBC Driver and the c-treeSQL Server.

CT_BOOL Correctly Maps to SQL_BIT with SQL Table Import

The c-treeSQL ODBC driver did not show YES/NO for boolean types but rather '1' or '0'. The c-treeSQL Import Utility, **ctsqlimp**, erroneously mapped a CT_BOOL to a SQL_TINYINT instead of a SQL_BIT. The **ctsqlimp** utility now correctly maps this value.

UPPER() Function Syntax Corrected

The following query statement failed with error **SQL_ERR_BADPARAM** (-20127),

```
"SELECT * FROM tab233 WHERE UPPER(Name) >= ?"
```

The UPPER function in the WHERE clause was not properly parsed as a statement parameter. The c-treeSQL parser engine was modified to handle this query construct.

c-treeSQL Log File no Longer Prevents Client Connection

A case was reported where the c-treeSQL Server was unable to open the log file *sql_server.log*. The c-treeSQL engine uses a standard OS **fopen()** function call to open the file and a low-numbered file descriptor (less than 256) was not available. An error, -1, was returned to the client connection. When the c-treeSQL Server fails to open the log file in this case, an error is no longer returned and operations are skipped on the log file that could not be opened.

Correct Linking of Tables in MS Access

When attempting to link a table into Microsoft Access using the c-treeSQL ODBC driver, the Error "Index already exists" occurred. Access reported this error as it called **SQLStatistics()** to get a list of the fields for each index, but the rows were not necessarily returned sorted by index leading to the error. The logic to return the rows has been adjusted to correct this error condition.

Dropping a Constraint Executed by a User Other than the Constraint Creator

The following statements executed properly while performed with an ADMIN user account:

```
create table ctreeuser.test (f1 integer);
alter table ctreeuser.test add constraint test_pk primary key (f1);
commit work;
```

```
alter table ctreeuser.test drop constraint test_pk;
rollback work;
```

However, if executing the following statement as another user:

```
alter table ctreeuser.test drop constraint test_pk;
```

the following error occurred:

```
error(-20029): Index referenced sys_001_000001155 not found
```

The index "sys_001_000001155" belonged to ADMIN (or the user who created the constraint) and this information was not properly stored in the internal system tables. These tables are now properly updated with the correct user information.

Correct Query Results Returned When Using an OR Conjunction

The following query returned the correct number of records,

```
SELECT DISTINCT (sees_q_as + '^' + q_name)
FROM ctree_db1 a, ctree_db2 b
WHERE (a.field1 = b.field1) AND (USER_ID = '1013184816_1738021448')
```

When the query was modified with an OR-ed conjunction and should either leave the query result untouched or add more rows, the query returned no rows:

```
SELECT DISTINCT (sees_q_as + '^' + q_name)
FROM in_q_users_db a, in_q_db b
WHERE (a.q_id = b.q_id) AND
      ((USER_ID = '1013184816_1738021448') OR (USER_ID IN (SELECT group_userid FROM
in_user_group_db WHERE user_id = '1013184816_1738021448')) )
```

It was found that a select query with an OR-ed sub-query predicate gave a wrong result as the c-treeSQL parsing engine did not replace the previous expression tree with the new expression tree after a tree transformation. The parsing engine has been corrected to properly handle this situation.

c-treeSQL ODBC SQLStatistics() Returns Correct Information

The c-treeSQL ODBC driver, when using the **SQLStatistics()** function, did not return any information about the indices. **SQLStatistics()** internally used a different statement handle and not the statement on which the call is made. Any binding of columns done by the application is not reflected on the cached statement handle. A change was made such that the data is now copied to the bound data variables.

Correct Number of Affected Rows Returned with SQLGetDiagField()

When the following c-treeSQL ODBC code was executed **SQLGetDiagField()** returns zero.

```
SQLPrepare(hstmt, "INSERT INTO test VALUES (?)", ...)
SQLBindParameter(hstmt, 1, ...);
SQLExecute(hstmt);
SQLGetDiagField(SQL_HANDLE_STMT, hstmt, 0, SQL_DIAG_ROW_COUNT, ...);
```

After a call to insert a row with a parameterized query containing a **LVARCHAR** field, **SQLGetDiagField()**, with no intervening calls, did not return the proper number of affected rows. The c-treeSQL ODBC driver did not properly update an internal data structure. This structure is now updated with the correct data.

It was also found when inserting literals into long fields, such as:

```
insert into long_table values ('literal string')
```

the c-treeSQL Server was passing the maximum available length rather than the correct length. The actual length of the literal is now properly passed by the server.

Proper Return of LVARCHAR Data With SQLExecDirect()

The **SQLExecDirect()** fetch method closed its cursor when it retrieved all of the rows it was called for, however, this did not allow **SQLGetData()** to retrieve the long data field content. An error code was returned when retrieving LVARCHAR data and this data was actually invalid, the result of garbage data.

The c-treeSQL Server was modified such that it does not close the cursor when a fetch involves long data fields thereby correctly retrieving all of the requested rows.

Note: This fix affects both the c-treeSQL Server and the c-treeSQL ODBC Driver.

Removed Required DSN Connection Strings for the Direct Link ODBC Driver

The ODBC call **SQLDriverConnect()** used with the direct link ODBC library returned an error if the connection string did not contain a DSN specification. For the c-treeSQL direct-link ODBC Driver this does not make sense. The code explicitly checked for the presence of DSN specification. Changed the code so that the check is not performed in case of direct link ODBC library.

Correct Array Handling with Prepared Statements in JDBC

The c-treeSQL JDBC driver returned a **NegativeArraySizeException** during a **prepareStatement()** method call with more than 20 parameter references. The JDBC driver did not properly resize an internal object to contain enough space for storing the description of all the parameters. The driver now properly resizes the object to dynamically handle any number of parameters.

More than 2000 Bytes Allowed with JDBC LVARCHAR Retrievals

It was not possible to retrieve more than 2000 bytes from an LVARCHAR type using JDBC. Additionally, the data returned did not appear to be correct. The 2000 byte limitation has been removed and this new behavior is per current SQL standards.

Proper Batch Resizing with the JDBC Driver

The following JDBC driver exception was reported while using a **prepareStatement()** and **addBatch()**:
`ctree.jdbc.DharmaSQLException: Row value out of range`

The error occurs after executing the batch and then executing a second batch using the same prepared statement. The JDBC driver did not properly resize an internal object used for batches after executing the first batch. This collection is now resized after a successful execution.

Proper Batch Additions with VARBINARY Fields

The c-tree JDBC Driver failed to properly handle VARBINARY fields in parameterized JDBC batches. The JDBC driver returned an underflow/overflow error when executing the batch and the VARBINARY field was of a decreasing size with each insert statement. The JDBC Driver failed to properly evaluate the column size with each insert, and this behavior has been corrected.

Correct JDBC Exception now Thrown when User Limit is Exceeded

When the user limit was exceeded while connecting to the c-treeSQL Server from the c-treeSQL JDBC Driver, it was possible to experience a “lock-out” condition until the application was restarted. The c-treeSQL Server correctly returned the proper state information, however, the JDBC Driver incorrectly threw a “null reference” exception. The correct error in this case is a **-17084** (Maximum Users Exceeded).

c-treeSQL Date Field Range

The following c-treeSQL query on a table containing a date field and an index on that field did not return the proper rows:

```
SELECT * FROM table1 tab WHERE tab.duedate >= '0001-01-01' AND tab.duedate <= '9999-01-01'
```

However, this one worked:

```
SELECT * FROM table1 tab WHERE tab.duedate >= '1700-03-01' AND tab.duedate <= '9999-01-01'
```

The c-tree Plus date field stores date as an unsigned 4 bytes integer value representing the number of days since 02/28/1700. Hence the first valid date is 03/01/1700. The algorithm converting the date into the 4 byte unsigned integer did not take into account negative numbers, hence the dates before 02/28/1700 result in a large value. The combination of the two conversions, (date --> integer --> date) results in a date different from the original one.

A check was added to limit the date range lower bound to 03/01/1700 in case of a c-tree Plus date type and 01/01/1900 for the c-treeSQL `TIMESTAMP` type.

Correct Join Handling in Queries

The c-treeSQL Server failed to execute a particular class of `SELECT` queries containing a join resulting in error **20008**. A particular optimization strategy failed to produce the correct output. The query optimization logic was adjusted to use an appropriate optimization for this type of query.

Proper Cursor Close with Stored Procedure Resultsets

c-treeSQL error **-20125** (no cursors available) was encountered after running a stored procedure performing a select statement with an empty resultset.

The c-treeSQL Server did not properly close the cursor when the resultset was empty. Explicitly closing the cursor internally now prevents this error.

c-treeSQL Server System Signal Handling

When `CONSOLE CTRL_C_ENABLE` is specified in the c-treeSQL Server configuration file a `<CTRL>-C` key combination or the `SIGINT` signal should begin a normal server shutdown process. It was found that the c-treeSQL Server ignored this signal as well as the `SIGQUIT` and `SIGTERM` signals. Proper signal handling has been enabled on the c-treeSQL Server such that correct operation is restored for this useful c-treeSQL Server configuration option.

Retrieval of `LVARCHAR` Fields Using the c-treeSQL ODBC Driver and ADO .NET

When reading a `LVARCHAR` field using ADO .Net and the c-treeSQL ODBC driver, a truncated result containing much less than the actual field content was returned.

The ODBC **SQLGetData()** function set a parameter, *StrLen_or_IndPtr*, to the number of bytes read rather than the number of bytes remaining to be read. This value was returned by the c-treeSQL Server. The c-treeSQL Server `LONG` type handling has been corrected to handle this condition. This also required a minor change to the c-treeSQL ODBC Driver.

Improved JDBC and ODBC Driver Handling of Empty Resultsets with Long Fields

It was found that a call to **ResultSet.close()** or **PreparedStatement.close()** resulted in an exception “cursor not open” when the query selected a Long `VARCHAR` field and there was no row matching the criteria.

When fetching records, the last fetch closes the cursor on the server side. In the case of long fields, though, the cursor stays open and the c-treeSQL JDBC or ODBC drivers knows to handle the cursor close depending on the long field existence; in some cases, the cursor remains open. It was discovered that when the result set was empty, the cursor was always closed, regardless of the existence of a Long field, and the explicit **close()** generated the exception. JDBC and ODBC driver implementation documentation states that this is an accepted condition, and the **close()** method should succeed regardless.

The c-treeSQL JDBC and ODBC drivers were modified such that a **close()** on an already closed Statement object is properly handled according to standard.

Proper Semicolon Syntax with ISQL

The c-treeSQL Interactive SQL utility, ISQL, would hang under a particular query when a semicolon was placed after an END statement. The syntax parsing has been corrected such that an error is now returned in this situation.

Eliminated Overflow/Underflow Errors Involving Joins

A query involving a join condition of two char fields of differing sizes fails with c-treeSQL error **2052** (overflow/underflow). The c-treeSQL engine has been corrected to avoid this case.

Corrected SELECT COUNT(*) from a Variable Length Table Without an Index

A variable-length data file with no index contains no count of the number of active records. In this case, the c-treeSQL Server must scan the variable-length data file, counting the number of active records. The function that scans the variable-length data file calls an internal function to retrieve the file control block for the specified data file. However, the file number passed to the internal function was a user file number, rather than a system file number and this was not properly handled. By correctly checking the file number passed, the c-treeSQL Server now correctly counts the records in the appropriate file.

Corrected JDBC -21043 Error

A JDBC exception (-21043, c-treeDB Error 4023: Table does not exist) was reported, along with bad content returned from the c-treeSQL interactive ISQL utility in a `LVARCHAR` field. Upon internal investigation, the two problems were found to be related.

When initializing the field handle to the `LVARCHAR` field, an internal c-treeSQL flag was improperly not set indicating the object is bound to the data. The c-treeSQL engine then lost information about the

field, resulting in a random table ID. This set the condition for the internal c-treeDB 4023 error (table does not exist). This error was returned to the JDBC client as c-treeSQL error -21043. This flag is now appropriately set correcting this error.

Proper LVC Update with the c-treeSQL JDBC Driver in Autocommit Mode

While updating a row containing a LONG VAR field with a prepared statement and autocommit enabled, exception -17057 was reported with the c-treeSQL JDBC Driver.

Internally, the JDBC driver attempts a commit operation (due to the autocommit behavior) while partially updating the row. This results in the current record locks to be released, which are expected in the second phase of the LONGVAR update. c-tree error **DADV_ERR** (57) is returned, and reported in c-treeSQL as -17057.

The c-treeSQL JDBC Driver now temporarily disables the autocommit mode such that the transaction is not automatically committed in the first phase of the update. An explicit transaction commit then properly completes the operation. This logic takes place only when autocommit is enabled.

Improved c-treeSQL JDBC Performance when Resizing the SQLDA Object

The c-treeSQL JDBC driver performance remained sub-par after previous modifications. A review of the logic determined that a resizing of the *SQLDA* object continued to take place, even in the case where the array remained of equal size. This behavior has been changed such that the object is resized only when the array resize is necessary.

Physical File Name Changed with c-treeSQL RENAME TABLE Statement

The c-treeSQL Server did not change the physical file name on disk when a `RENAME TABLE` statement was given. Rather, the c-treeSQL Server updated the system table entry containing only a link to the physical file.

This behavior has been changed such that by default the `RENAME TABLE` statement also renames the table (file) on disk. The new physical table name follows the c-treeSQL convention and may have a prepended owner name attached.

Increased Number of Network Backlog Users Allowed

It was determined that the cause of a c-treeSQL 30034 error was due to the number and speed of database logins. In this case, after 100 attempted user connections the network timeout error occurred. The source of the limitation was found to be the Winsock `listen()` backlog size, which was a default of 50 for c-treeSQL connections. A TCP/IP `SOMAXCONN` value of 1024 (the largest value Windows XP supports) has been implemented minimizing this congestion.

Note: This value is operating system dependent, and not all operating systems will support this value.

Deferred Key Add Operations For Update Of Transaction-Controlled Unique Index

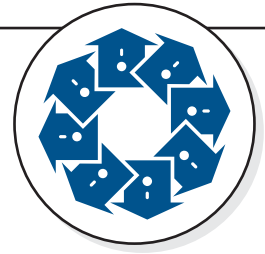
A c-treeSQL update operation that increments the value of a field on which a unique key is defined could fail with a duplicate key error (**KDUP_ERR**, 17002) even though the resulting set of key values did not contain duplicates. The following c-treeSQL commands demonstrate this situation:

```
CREATE TABLE updtest (page_no INTEGER)
CREATE UNIQUE INDEX updtesti ON updtest (page_no)
INSERT INTO updtest VALUES (1)
INSERT INTO updtest VALUES (2)
INSERT INTO updtest VALUES (3)
UPDATE updtest SET page_no = page_no + 1 WHERE page_no >= 1 AND page_no <= 3
```

c-treeSQL processes this update operation by reading each record matching the criteria and updating it. When the record having *page_no* of 1 is read and updated, its new value of 2 conflicts with the second record's *page_no* value of 2.

A record update that changes a key value performs two key operations: deleting the original key value and adding the new key value. The duplicate key error that occurs in this situation can be avoided by causing a record update operation to delete the original key value and temporarily skip the addition of the new key value. If all record updates that are performed when processing a c-treeSQL update operation are done in this manner, then the new key values can be added after all the updates have been performed, and no key duplicate errors will occur due to the attempted addition of an updated key value that matches one of the original key values that is being changed by this update operation.

This deferred key add operation is now supported by c-treeSQL to avoid the **KDUP_ERR** in the described situation.



c-treeACE Server

3.1 Critical Issues



c-tree Server Hang Corrected

Occasionally, a c-tree Server hang was reported by a customer. An internal server function was found that could generate a recursive call deadlocking on an already held semaphore. Proper detection of this semaphore is now done avoiding the recursion, and more importantly, greatly increasing the stability of the c-tree Server.

Improved Shared Library Support on Linux

The c-tree Server on Linux experienced a segmentation fault in a system function call after loading a user-created shared library.

The use of the *-static* link option when compiling the c-tree Server could result in a situation where the server loads a shared library that has external shared library dependencies with conflicts. Examples include a *CT_USER* shared library or callback shared library, if these libraries reference external shared libraries. The server and the shared library could potentially use different copies of the same library, thereby producing an inconsistency that resulted in a crash. Compile time settings have been modified to prevent this situation.

Fatal Log Write no Longer Hangs Server at Shutdown

If the c-tree Server encounters a fatal error writing to the transaction logs, it begins a controlled shut down, as it cannot continue operations without proper transaction logs to guarantee automatic recovery. It was found that this shutdown sequence could hang in some instances. The server status log showed the following messages:

```
Thu May 12 09:36:43 2005
- User# 43      043 M0 L52 F2 P4x (recur #1) (uerr_cod=152)
Thu May 12 09:36:50 2005
- User# 43      System Log thread still running
Thu May 12 11:04:29 2005
- User# 18      tflstt loop alert
```

```
Thu May 12 11:04:29 2005
- User# 26      tflstt  loop alert
Thu May 12 11:04:29 2005
- User# 24      tflstt  loop alert
```

The status log messages revealed the server shut down because the server failed to extend a transaction log while committing a transaction. A server process stack trace showed the thread that encountered the fatal error had been deadlocked. The variables leading to this condition have been corrected.

CTSTATUS_SIZE and DIAGNOSTICS LOWL_FILE_IO Configuration Options no Longer Hang Server

It was found the c-tree Server hangs during normal operation when using the `CTSTATUS_SIZE` and `DIAGNOSTICS LOWL_FILE_IO` keywords.

`DIAGNOSTICS LOWL_FILE_IO` instructs the server to log messages to `CTSTATUS.FCS` when file open/create/delete/rename operations fail. `CTSTATUS_SIZE` limits the size of the server status log. Before creating a new transaction log, the server attempts to open the log. As expected, the log does not exist and the server logs a diagnostic message to the status log indicating that the open failed (because `DIAGNOSTICS LOWL_FILE_IO` is enabled). This recursive semaphore request deadlocked the server. A change to the semaphore request logic corrects this server hang.

MEMORY_MONITOR Option no Longer Crashes Server on Windows 2000

An incorrect print format specifier on Windows 2000 caused the c-tree Server to crash when writing messages to the memory monitor window when the `MEMORY_MONITOR` c-tree Server configuration keyword was specified. The correct print function is now used when writing these messages.

Server no Longer Crashes when Passed a datno from BlockingISAMRead

The c-tree Server experienced a hard crash when a user passed in a valid `datno` in a `BlockingISAMRead()` call. An uninitialized variable was found to cause the problem. The correct assignments are now made resolving this issue.

Maximum Connected Client Limit no Longer Terminates the c-tree Server

The c-tree Server had shut down unexpectedly during normal operation with the following messages found in the status log, `CTSTATUS.FCS`:

```
Wed Oct 19 14:17:58 2005
- User# 00011NewUser: Unable to create an instance of a named pipe
Wed Oct 19 14:17:59 2005
- User# 00011Could not initialize the server.
Is another server already running?
Is a detached client still running?
Wed Oct 19 14:18:00 2005
- User# 00011O11 M1 L42 F0 P54x (recur #1) (uerr_cod=84)
```

While checking the new connection against the number of allowed connections, the server improperly interpreted an internal return code as an irrecoverable startup error, rather than a common run-time error. A new flag was added to the logic to properly process the error, preventing an unnecessary server shutdown.

“Abort node error” When Closing PREIMG files

While closing files for a client that has abnormally disconnected, the c-tree Server reported an “Abort node loop” error on a *ctPREIMG* index and marked the index corrupt, requiring the index to be built. In one case, the message was accompanied by E69 messages, in which the file control block’s member number and the index buffer’s member number differed.

It was discovered the files were created without 6-byte transaction number support and the c-tree Server’s transaction numbers had exceeded the maximum supported transaction number for a non-6-byte transaction capable index. The c-tree Server’s key add and delete operations contained a check for an add or update on a non-6-byte transaction capable index when the transaction number exceeds the maximum supported transaction number for this type of file, but the check only applied to *ctTRNLOG* files. As a result, the c-tree Server inappropriately permitted key add and delete operations on non-6-byte transaction files in this situation. Internally used bits of the transaction numbers became overwritten resulting in the “Abort node loop” errors.

The transaction number check on key add and delete operations was corrected so that it also applies to Pre-Image files. The new behavior that results from this change is that a key add or delete operation will fail with error **OTRN_ERR** (534, transaction number overflow) in this situation. Anyone encountering this problem should update to a c-tree Server with this fix applied and should clean their transaction controlled index files using the **ctcIntrn** utility and should delete the transaction logs after cleanly shutting down the c-tree Server, so that the c-tree Server restarts its transaction numbering.

3.2 Serious Issues



WRITE_ERR Notification

The potential possibility of a disk write failure may go unreported with the c-tree Server. A particular scenario has been identified where in the event of a data or index file write failure, the c-tree Server continues operations, and assumes 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.

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
```

An additional new server configuration keyword will cause the contents of the write request to be appended to the *WRITE_ERR.FCS* file.

```
DIAGNOSTICS WRITE_ERR_DUMP
```

If the write is for more than 32K bytes, then only the first 32K bytes are output to the file.

Correct Path References with the Novell NLM

The NLM c-tree Server failed to open a c-tree data file and returned error **FCRP_ERR** (14) when one client opened a c-tree data file using no path, or an absolute path, and then another client opened the file using a relative path. The server converts the relative path to a full path and compares the two paths. But the Netware function used by the c-tree Server to convert a relative path to a full path did not properly remove relative path references (*.*) and (*..*) from the directory name. The server found the two paths did not match and treated them as two separate physical files. When the server attempted to open the file the second time, it found the update flag was set and failed with **FCRP_ERR** on the file open. The c-tree Server now converts relative path references in the full file path to the appropriate full path specification. The server also correctly removes occurrences of *.* from the path and replaces occurrences of *..* with the appropriate parent directory name.

Record Conversion Buffer Overflow Fixed

An internal conversion routine could attempt to swap byte ordering on a field that was not completely contained in the allotted buffer. A typical situation for this problem is the use of the **xxxVREC()** routines which permit partial record reads. If the buffer was not large enough to hold the entire record, we return a partial record. The code to convert byte order checked for buffer overruns but not for binary fields that may be truncated at the end of the buffer. The logic to determine the length of these fields has been changed to properly handle this situation.

Improved Dynamic Dump of HUGE files for c-tree Servers on Linux

On Linux operating systems, the c-tree Server's dynamic dump did not include files over 4 GB in size in the dynamic dump backup even though the dump script specified a wildcard filename that matches the name of the file. If the file was explicitly listed by name in the dynamic dump script, it was included in the backup. The correct filesystem calls are now used to ensure that all files, including *HUGE* files (files over 4Gb), are included in the c-tree Server dynamic dump for this platform.

Dynamic Dump Includes Matching Filenames with Wildcards and LOCAL_DIRECTORY Configuration Keyword

On Unix systems, when the c-tree Server's dynamic dump script contains a filename with a relative path and wildcards in the filename, and the c-tree Server is configured to use the `LOCAL_DIRECTORY` keyword, the dynamic dump did not find files matching the wildcard specification.

For example, if using `LOCAL_DIRECTORY data/`, and the file `data/accounts/acc01.dat` exists, and the dump script specifies the filename `accounts/acc01.*`, the dynamic dump did not back up the file `data/accounts/acc01.dat`. If the file was `data/acc01.dat` and the script specifies `acc01.*`, the dynamic dump did back up the file.

The function used to find files matching a wildcard specification on Unix systems failed to properly prepend the `LOCAL_DIRECTORY` path when specified. The function only prepended the local directory name when the filename did not include a path.

The function was corrected to prepend the `LOCAL_DIRECTORY` directory name to the filename regardless of whether the filename contains a path.

Proper Server Shutdown with the ADMIN_ENCRYPT Configuration Keyword

It was found that the c-tree Server could fail during shutdown when the server configuration keyword `ADMIN_ENCRYPT` was specified and a component of the `FAIRCOM.FCS` file still required flushing during shutdown. The server then fails to start the next time as automatic recovery fails to occur.

Internal initialization in the checkpoint logic was modified to determine if the encryption routine was properly initialized. This now allows a proper checkpoint at server shutdown to take place with the encrypted contents of the c-tree Server user and group information file.

Compacting HUGE Files with the c-tree Server Properly Returns all Records

A case was discovered where a *HUGE* c-tree data file compacted by the c-tree Server was missing records that were present in the original data file. When a data file larger than 4GB was compacted, it was found that an inappropriate file offset value (the high word portion) was used internally resulting in a failed operation. Copying of data was then prematurely stopped, however, the finalizing operations continued, creating a properly constructed compacted file lacking a complete copy of all the data records from the original, now deleted. The proper offset values are now passed avoiding this data loss.

Corrected Connection Memory Leak

It was found that c-tree Server memory usage unexpectedly increased over time as clients connect and disconnect.

When closing a superfile, the c-tree Server failed to free a small buffer used with the directory index. This omission caused a memory leak when a client connected, as when the client connected, it opened the superfile *FAIRCOM.FCS*, authenticated the user, and closed the superfile. This memory is now properly freed in all cases preventing the observed leak.

3.3 Other Issues

c-tree Server Timeout Feature on Shutdown

It was possible for the c-tree Server to take a very long time to shut down. It was determined the TCP/IP connection queue could be filled with client connect requests, and a system dependent timeout could delay the processing of the internal server shutdown socket connection.

The socket timeout feature has been enabled for the c-tree Server and sets a ten second timeout limit on the connect call made by the server thread that is shutting down the c-tree Server. The expectation is that if the connect call times out after ten seconds, there is not a communications listener thread in an `accept()` call that needs to be terminated.

Note: This modification also turns on the client socket timeout feature at compile time, but the client timeout is disabled by default and is enabled by setting the `ctsockettimeout` c-tree global variable to the desired timeout value.

File System Cache Now Flushed Before Physical Close

It appeared possible in some situations, 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-tree Server transaction control system would have no knowledge of this. Extra protection has been added to ensure that this situation did not occur.

A new behavior, `ctBEHAV_FLUSHonCLOSE`, will cause a system cache flush before physically closing transaction logged files or write-thru files. This behavior can be disabled by adding `COMPATIBILITY NO_SYS_FLUSH_ON_CLOSE` to the server configuration file.

Dynamic Dump Detects Incorrect Header Values for non-HUGE Files

The c-tree Server's Dynamic Dump operation failed to terminate at the proper end of a file. When it encountered a non-*HUGE* index file with an extended header unexpectedly containing non-zero values in the `numrec2` and `phyrec2` members of the extended header. These values should be zero for a non-HUGE file.

During a file open the Dynamic Dump now detects any unexpected header values and checks if it is safe to reset the `numrec2` and `phyrec2` values to zero. If it is safe, the values are reset. If not, error **SEQ8_ERR** (689) is raised. In both cases, an explanatory message, including the file name, is sent to `CTSTATUS.FCS`.

Note: Just because the Dynamic Dump resets the header values does not guarantee the header will be correct on disk. The file must be updated for the header to be written to disk. If it is not updated, the same action/message will be taken the next time the file is opened.

Update of Internal Unique File ID Corrected

The following entry in a `CTSTATUS.FCS` log was brought to our attention:

```
WARNING: unique file ID delete failed: 8770
```

A check for a change in unique ID mistakenly found the old and new IDs the same. c-tree now correctly tests the old and new IDs to see if the unique file ID index must be updated.

Correct Key Transformations with the Unicode Enabled Server

When **TFRMKEY()** was called by a c-tree client for a Unicode key segment the c-tree Server produced an incorrectly-transformed key value. It was found that the source and target buffers passed to the c-tree Server's internal key transformation function overlapped. The key transformation function already checked if the source and target buffers had the same address and took appropriate measures. Now, additional checks are done to examine if the source and target buffers overlap, and to handle this case appropriately, thus avoiding this error.

VIRTUAL Open of Segmented File Corrected

The following error message was noted in a *CTSTATUS.FCS* file:

```
vtclose failure: LOC 2: TOT 1: CHN 0 NOV 0 LOK 0 NFD 0 IOS 0
```

An attempt to open a segmented file with a file mode including *ctVIRTUAL* caused an internal state variable to be incremented and never correspondingly decremented. A segmented file cannot be opened virtually, and by the time this had been determined the internal variable had already been incremented. The subsequent close of the file did not decrement the variable because the close saw the permanent status. The open file logic was adjusted to cope with this situation.

“Improper Shutdown Detected” Dialog Suppressed

In some cases, the c-tree Server displays dialog boxes containing error or warning messages. The configuration option `CONSOLE NO_MESSAGEBOX` causes the c-tree Server to suppress the display of these dialog boxes. But the dialog box the c-tree Server displays at startup when it detects an improper server shutdown was not suppressed by this keyword. With this modification the c-tree Server will also suppress the improper server shutdown dialog box when the `CONSOLE NO_MESSAGEBOX` server configuration keyword is specified.

Proper Handling of Log Template Rename Failures

The c-tree Server directory could contain an unexpectedly large number of log template (**.FCT*) files. Normally when log templates are in use, only `LOG_TEMPLATE+1` log template files exist.

If the c-tree Server fails to rename a log template, or if the header read or write on a log template file fails, the server does not use the log template and instead creates a new transaction log. In these cases, simply delete the log template file that is not going to be used by the server.

SNAPSHOT Negative Statistics Output Corrected

Some of the c-tree Server *SNAPSHOT* statistics sometimes displayed negative values. Particular examples include:

```
average commit delay (usec)  
average log save time (usec)
```

An overflow condition was identified and the correct values are now output retaining their precision.

Correct Zero Timeout Handling with `ctSysQueueRead`

A call to `ctSysQueueRead()` with a zero timeout parameter did not return if the queue was empty, and the server begins to perform a fairly tight loop consuming excessive CPU cycles.

`ctSysQueueRead()` was designed to be interruptible if it is called with a non-zero timeout, however, the zero timeout condition was not properly handled. It failed to block on the queue control structure when the queue was empty, and kept looping to check the queue status. The zero timeout case (as well as the non-zero timeout case) must attempt to block on the queue control structure.

If the queue is empty and the timeout is zero, the blocking call will now immediately return with a `NTIM_ERR` (156) which is returned by the `ctSysQueueRead()` call.

Correct Return Values of System Queue Functions

The functions `ctSysQueueOpen()`, `ctSysQueueMlen()`, and `ctSysQueueCount()` are documented as returning error codes as negative values, but were found to return error codes as positive values. The single-entry point feature of c-tree Plus failed to negate the error code for these functions as expected. These functions have been added to the cases considered in which the error code is to be negated.

Update when a `SCHSEG` key Segment is Defined on a Unicode Field

A record update on a file containing a `SCHSEG` key segment defined on a Unicode field without specifying the `UNCSEG` mode failed with error `USEG_ERR` (707 , no segment definition).

Although the `UNCSEG` mode was applied to `SCHSEG` key segments defined on fields having Unicode data types, the code leaves the extended key segment handle set to zero. The extended key segment handle of zero caused an internal call to fail with error `USEG_ERR`. When the low-level key assembly function `ctasskey()` automatically applies the `UNCSEG` mode to a `SCHSEG` key segment, we now set the extended key segment handle to -1 such that the Unicode segment definition data for the key segment is initialized.

Automatic Switch to `O_SYNC` in Case of Direct I/O Error

On Solaris, the c-tree Server implements synchronous system disk writes using direct I/O. However, not all file systems support direct I/O. Although the direct I/O system call returns an error when called for a file system that does not support direct I/O, the server did not take corrective action to ensure that the file's writes were written to disk (avoiding the file system cache).

Now, if the system call to enable direct I/O for a c-tree log/data/index file fails, the server automatically switches the file into synchronous mode using the `O_SYNC` file access mode. The code now checks the return code on all calls to `ctsyncx()` and `ctsync()`, so if setting a file into `O_SYNC` mode fails, the server will return an error for the write operation.

The server logs the following message to `CTSTATUS.FCS` when it fails to enable direct I/O on a file (after which it switches to `O_SYNC` mode for that file):

```
ctsyncio: Direct I/O request failed on fd <descriptor> for file <filename>
```

The server logs the following messages to `CTSTATUS.FCS` when it fails to enable `O_SYNC` I/O on a file (the first message indicates that the file status flags could not be retrieved; the second message indicates that the file status flags could not be updated to include `O_SYNC`):

```
ctsyncio: Failed to retrieve file status flags on fd <descriptor> for file <filename>
ctsyncio: Synchronous I/O request failed on fd <descriptor> for file <filename>
```

Correct SIGNAL_READY Syntax with Long Paths

On Windows systems, when the c-tree Server configuration file specifies an executable file name containing spaces for the `SIGNAL_READY` option, the server fails to run the specified executable file. The c-tree Server now converts this file name to the Windows short name to avoid problems with included spaces in the pathname.

Invalid Space Management Index Root After Recovery

An error `IEOF_ERR` (519) was reported in production after automatic recovery occurred.

The space management index of a variable length data file was not transaction controlled. If a variable length data file was written to during automatic recovery, a flag was set resulting in the space management index to be reset as empty and the c-tree engine then reconstructs the index in the background. This particular case involved a variable length data file whose logical file size was reduced during an automatic recovery file open, and no other write was issued. The smaller size invalidated the index root position, however, no flag was set to indicate the space management index was no longer viable.

A detection was put into place such that if a variable length data file header is updated during an automatic recovery open, then, in addition to checking the write flag, a check if the root is no longer viable is done. If either a write has been issued or if the root is positioned after the logical end-of-file, the space management index root is reset to zero.

TMPNAME_PATH for Rebuild Sort Work Files now Operates as Expected with the c-tree Server

Sort work files are used during a rebuild operation and take up additional disk space. When using the c-tree Server, it is possible to specify an alternate location for these files with the `TMPNAME_PATH` server configuration keyword. It was determined that this path was not updated and remained the value set at server compile time. This behavior has been corrected such that the sort work files are now created as expected in the `TMPNAME_PATH` directory as specified in the server configuration file.

Correct GTEKEY Retrieval with 64-bit Memory File Key Values

When `GTEKEY()` was called for a 64-bit memory file on a low/high byte ordered system, the c-tree Plus logic that converted 64-bit memory file record addresses incorrectly considered the memory address as a high/low value. `GTEKEY()` failed to return a matching key value in this situation. The byte order is now properly considered in the case of 64-bit memory files.

Read-Only Dynamic Dump Script Files

It is useful to mark Dynamic Dump script files as read only to prevent an accidental overwrite. If the c-tree Server attempted to open a dynamic dump script file that was marked read-only, the server failed to open the file. The logic used to open the file has been adjusted to allow the file to be opened in this case.

Correct Index Recovery After a Checkpoint Splits a Transaction Commit

After automatic recovery, it was found that a key value was missing from an index. Missing key values after recovery could occur when a transaction commit writes the `TRANEND` log entry and is then interrupted by a checkpoint before completing the commit. For the problem to occur for a particular

index, the split transaction commit must have had all of its updated nodes for the index flushed to disk. In this case the checkpoint contains conflicting information:

1. The transaction must be redone (at least with respect to data cache pages).
2. The index may need some nodes to have their update operations undone (because of abort node list entries).

The first of these states is correct and the second was wrong. The problem has been corrected during the recovery phase (which occurs infrequently) rather than each checkpoint.

SEQ8_ERR Added to System Message Management File

The system message management text input file, *ctsysm.cfg* has been updated. **SEQ8_ERR** (689) was added to a *CTSTATUS.FCS* message when values cannot be safely reset.

Additional Errors Added to Server Shutdown Exclusions

This change will allow a c-tree Server better handle a failed update to the space management index of a variable length data file that occurs when committing a transaction. Prior to this change, if the space management index update failed with any of the following errors the c-tree Server shut down:

- **EVAL_ERR**
- **REXT_ERR** (748)
- **TABN_ERR** (78)
- **BNOD_ERR** (69)
- **IEOF_ERR** (519)

These errors are now excluded for potentially shutting down the server and the server instead logs the following message to *CTSTATUS.FCS*:

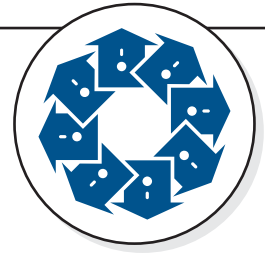
```
"Could not update space management index during end-of-transaction clean up"
```

The name of the variable length data file whose space management index could not be updated is included along with this message. The transaction commit then continues as usual.

c-tree Server Defunct Process Cleanup with SIGNAL_READY Handling on Unix Systems

The c-tree Server provides optional startup and shutdown signals, via the `SIGNAL_READY` and `SIGNAL_DOWN` c-tree Server configuration keywords, allowing an external process to perform pre-and/or post-operation activities. It was found on Unix systems, that the process created by the `SIGNAL_READY` server configuration option remained in the process list as a defunct process until the c-tree Server process terminated. Furthermore, when using the c-treeSQL Server, the processes terminated normally as expected. This behavior has been corrected and is now consistent with both servers.

The c-tree Server executes the new process with the standard `fork()` and `exec()` function calls to create a new process. To clean up the child process resources when the child process exits, the parent process (the c-tree Server in this instance) must either call `wait()` to wait on the child process to terminate or it must ignore the `SIGCLD` signal. The c-treeSQL Server already ignored the `SIGCLD` signal, however, the c-tree Server did not. The c-tree Server now ignores the `SIGCLD` signal on Unix systems, and any child processes should terminate as expected.



c-treeACE for .NET

4.1 New Dispose() Method for c-tree Plus for .NET

Instantiating multiple c-treeDB objects (*CTSession*, *CTTable* etc.) and allowing these objects to go “out of context” (losing all of the objects’ references) resulted in an Unhandled Exception in low level code.

When the .NET Garbage Collector (GC) begins to collect unused objects, it initiates a new thread that may interrupt a c-tree operation using a specific context. It then begins destroying c-tree objects that may be in a different c-tree context. When normal operations resume, the active c-tree context may be different from the original one

The new behaviour is to avoid DbLayer object collection by the GC. This is achieved by removing the class destructors. The objects must be manually destroyed, and the resources freed by the programmer using a new method called **Dispose()**.

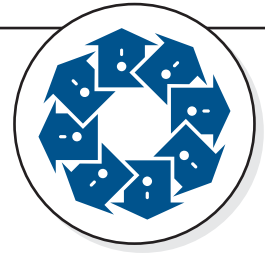
4.2 Resolved Conflict Between ctNUMBER and CTNumber class

In “case insensitive” environments (i.e., Delphi) there was a conflict between the *CTNumber* class and the *ctNUMBER* structure, as well as between the *CTBlob* class and the *ctBLOB* structure. To avoid this conflict, the two structures have been renamed as follow:

- *ctNUMBER* renamed to *ctdbNUMBER*
- *ctBLOB* renamed to *ctdbBLOB*

4.3 Correct Return Types for GetFieldAsFloat()

ctdb.GetFieldAsFloat(...) and **CTRecord.GetFieldAsFloat(...)** were incorrectly returning a float instead of a double. They now properly return a double value.



c-treeDB

5.1 Critical Issues



Memory Leak in `ctdbFreeRecord`

A perceived performance issue was reported when using the c-treeDB API. Using Shark, an Apple profiling tool, to analyze the behavior it was identified what could be context resources not being freed in c-tree Plus. The `ctdbClearRecord()` function call was found to use about 50% of the main thread's processing time, nearly all of which seem to be devoted to obtaining a context for the record. An improper check was made to determine a valid context ID and this check has been corrected.

Disappearing Records with `ALTER_TABLE` Resolved

A customer began to see all the records in some of their data files disappear after a `ctdbAlterTable()` operation. `AlterTable (CTDB ALTER FULL)` using `CTDB ALTER NORMAL` or `CTDB NORMAL INDEX` does not remove the records, but neither do they repair the index file. Internally, c-treeDB received an `INOT_ERR` (101) error on the call to `FirstRecord()` as the index is traversed and thus assumed there were no records. This issue has been fixed by setting the default index to `CTDB_DATA_IDXNO` which forces a table scan using the data records instead of an index scan.

5.2 Serious Issues



Dictionary Operations Now Properly Keeps All Locks

It was possible in some situations that record locks were lost in a table after opening a database in a second *CTTable* object.

The c-treeDB dictionary lock and unlock routines were modified to account for the case where the *KEEPLOCK* mode is active. The new dictionary code releases only the locks associated with the dictionary table. An extra check in `ctdbBegin()` was added to check for `IPND_ERR` (112) errors returned by c-tree `TRANBEG()`. If `TRANBEG()` returns `IPND_ERR`, we now suspend locks and call `TRANBEG()` again to restart the transaction.

ctdbBegin no Longer Fails When CTKEEP_MODE is Active

The c-treeDB C++ API Developer's Guide for `CTBase::Begin()` states:

The `Begin()` method does not lock or set the flags to lock the records. To request the lock of the records to be updated inside the transaction, the `Lock()` function or `CTRecord::LockRecord()` method should be used.

However, c-treeDB arbitrarily set the mode to `(ctTRNLOG | ctENABLE_BLK)`. Previously, this was not a problem as the only `Commit()` mode available was `ctFREE`, which would reset the ISAM LOCK flag. Ultimately, an application could receive an unexpected error `TTYPE_ERR` (99) on an update, after what appears to be a successfully begun transaction.

FairCom has added a `SetKeepLock()` session method which allows you to supply one of several modes to be passed to the commit.

Incorrect Index File Extension with Alter Table Corrected

When an application calls the c-tree DB alter table function with a mode of `CTDB_ALTER_INDEX`, the index file extension in the data file's *IFIL* resource may be set to an unexpected value. The result is that a subsequent call that used the index extension may fail or may produce unexpected results. For example, a subsequent alter table call may fail with error `KCRAT_ERR` (16) if the index extension contains characters that the file system does not allow in file names, or the index may be created with an unexpected extension.

An uninitialized variable was found to be the problem. Now, the rebuild function always initializes the index extension, avoiding this error.

Proper AtlerTable() of non-HUGE Files

A call to the c-treeDB alter table function for a non-huge file failed with error `MAPL_ERR` (653) in standalone multi-user mode.

The c-tree DB alter table function, called with a mode of *CTDB_ALTER_INDEX*, rebuilds the indexes for the specified file. When c-tree evaluated how to treat resources in an index about to be rebuilt, the file-size-checking code did not properly handle the high order words of the file control block (*FCB*) for a non-huge file. The file-size-checking logic now changes the high order words of the *FCB* only when the file is huge.

c-treeDB Path Prefix Now Properly Constructed

A customer reported a bad path prefix was constructed in a multi platform environment; specifically, a Windows client connecting to a Linux server. The path prefix code in c-treeDB used a path separator macro instead of an available utility function for this purpose. This utility function returns the correct path separator information under all operating circumstance.

ctdbWriteRecord no Longer Fails when Unicode Segment is Defined

ctdbWriteRecord() failed with error **USEG_ERR** (707, no segment definition) when an index segment was defined for one of UTF-16 field types *CT_UNICODE*, *CT_FUNICODE*, *CT_F2UNICODE* and *CT_2UNICODE*. This issue was fixed in c-treeDB code by adding a *UNCSEG* segment mode to index segments with an underlying Unicode field type.

c-treeDB Header and IFIL Modes now Match After Alter Table

It was reported that after an alter table full rebuild was performed, the data and index file mode stored in the header of the file was different from the data and index file mode stored in the *IFIL* structure. The alter table logic was fixed to update the *IFIL* structure data and index file modes to the same modes stored in the header of the data and index files.

Table Open with CTOPEN_READONLY Disallows Record Add and Update

If a table is opened with *CTOPEN_READONLY* mode, any record add or update operation should fail with error **SWRT_ERR** (458, write permission not granted). Tests have shown that c-treeDB tables with *CTOPEN_READONLY* mode allowed record add and update operations. Further tests showed that *CTOPEN_CHECKREAD* and *CTOPEN_CHECKLOCK* modes were not operating correctly either.

ctdbOpenTable() and **CTTable::Open()** were updated to ensure that the modes *CTOPEN_READONLY*, *CTOPEN_CHECKREAD*, *CTOPEN_CHECKLOCK* will trigger the correct c-tree ISAM modes when opening a table: *ctREADFIL*, *ctCHECKREAD* and *ctCHECKLOCK*.

5.3 Other Issues

c-treeDB now Removes .FCZ Files After Alter Table Operation

c-tree .FCZ files are created when a file using the *ctRSTRDEL* (restorable delete) filemode is deleted. This filemode allows a rollback to undo a file delete operation. An .FCZ file is also created when a *ctTRANDEP* (transaction dependent file) is deleted but the transaction is not yet committed.

In either case, the .FCZ file is simply a renamed copy of the original file. A c-treeDB alter table operation could cause .FCZ files to remain as the following example scenario demonstrates:

```
create table t1 (f1 integer);  
alter table t1 add (f2 integer);
```

Two temporary files are created: lets call them *ZA1.FCZ* and *ZA2.FCZ*

```
create table t2 (f1 integer);  
drop table t1;
```

Now two additional temporary files are created: lets call them *ZB1.FCZ* and *ZB2.FCZ*

```
commit work;
```

ZA1.FCZ and *ZA2.FCZ* are removed, but *ZB1.FCZ* and *ZB2.FCZ* remain

Improper consideration of the restorable delete filemode was found in an internal c-treeDB function when updating the file mode. This flag is now set only when required, and the .FCZ files are cleanly removed after the alter table operations.

c-treeDB Index Name can now be Changed

An index name could not be changed once an index was created. Index names were introduced with the addition of the c-treeSQL Server. Since c-treeSQL only adds or deletes indexes, no provisions were made to allow an index name to change once the index was created. Index names are also maintained by the c-treeDB database dictionary, as we need to relate an index *uid* (unique identifier) with an index name. The initial implementation of **ctdbSetIndexName()** generates error **CTDBRET_ISACTIVE** (4011) error if you try to call this function on a index handle of an existing index.

The c-treeDB implementation was changed to allow an index name to be changed, including the name of an existing index. For an existing index, you can change the index name by calling **ctdbSetIndexName()** function and then calling the **ctdbAlterTable()** function to perform the change.

c-treeDB Now Returns Specific Errors for Duplicated Index Names

The c-treeDB **ctdbAddIndex()** and **ctdbSetIndexName()** functions were changed to return specific error codes when an index name is duplicated. There are two specific return codes:

CTDBRET_INDEXEXIST (4051) - Index name already used in table.

CTDBRET_INDEXDUPNAME (4093) - Index name already used in database.

These error codes will be returned by the functions adding or setting the index name, instead of waiting until the table is created or altered to check if the names are duplicated.

Import of CT_DATE Field with Index Segment Mode of SGNSEG

CT_DATE fields caused a failed import of existing c-tree tables into c-treeSQL. The *CT_DATE* field index had a valid segment mode of *SGNSEG*, correct for legacy c-tree date types based on *DATET*, a

signed long. When the tables were imported, error **CTDBRET_INVSEGMODE** (4047) occurred as **ctdbAddSegment()** expected *CT_DATE* type segments to carry a mode of *INTSEG* (unsigned int).

CT_DATE and *CT_TIME* field types are now allowed to be used as valid index segments when the segment mode is *CTSEG_SGNSEG*.

ctdbGetRecordBuffer() Buffer Return Corrected

In the case of a variable length file, the **ctdbGetRecordBuffer()** function sometimes returned an invalid buffer when a variable length field was updated. When **ctdbGetRecordBuffer()** was called on a variable length record, after updating a variable length field, a buffer could be returned which did not correspond to the actual record buffer that should be stored in the c-tree file. The logic use to internally reorganize the record buffer now returns the correct buffer.

File Extension Detection Logic Now Working Correctly for Empty Extensions when Importing Files with ctsqlimp

It was found in c-treeDB that an empty file extension was not properly recognized when importing a table with the c-treeSQL Import utility, **ctsqlimp**. The table was imported with an empty table name. This behavior has been changed such that these tables are properly imported and recognized.

ctdbGetFieldNumber Does not Properly set the Error

When calling **CTTable::GetFieldNumber()** with a field name that does not exist in the table, the function throws an exception with an incorrect, random error code. An internal call failed to properly set the error code, and this is now properly set with c-treeDB.

Owner Name Case Handling Corrected in c-treeDB

Trying to perform an alter table twice on a table created with c-treeSQL resulted in an error **INOT_ERR** (101). The **ctdbSQLLinkTable()** function was not properly converting the owner name to lower-case before deriving the table name with c-treeSQL, hence importing the table with a wrong name. The second time the alter table is called, the logic cannot find the table in the c-treeSQL system tables because of the wrong name. This case handling has been corrected, avoiding error **INOT_ERR**.

Existing Databases Properly Checked when calling ctdbAddDatabase

The **ctdbAddDatabase()** function did not check if a database dictionary actually existed before adding the database name to the session dictionary. The **ctdbAddDatabase()** function was modified to check if a database dictionary actually exists before adding a database name to the session dictionary. In this case error **CTDBRET_NOSUCHDATABASE** (4094), database does not exist or not found, is returned by the **ctdbAddDatabase()** function.

Correct Alter Table With a CT_FSTRING Field Length of 256

An issue arose when attempting to alter the length of a fixed length string field in a table with a *CT_FSTRING* length of 256. A fix was applied to the **_ctdbRebuildFull()** function to ensure a buffer large enough is allocated if the string field length is exactly 256 bytes.

CT_CURRENCY now Valid for SNGSEG Segment Mode

When linking tables containing *CT_CURRENCY* fields and a segment mode of *SGNSEG* (signed binary segment) with the c-treeSQL Import Utility, **ctsqlimp**, error **CTDBRET_INVSEGMODE** (4047, invalid segment mode) was observed. This problem has been fixed with a test for the *CT_CURRENCY* field type and *CTSEG_SGNSEG* segment mode in **ctsqlimp**.

Proper Connect to a Database Created Inside a Transaction

If a database was created by calling **ctdbCreateDatabase()** inside an active transaction, then a database connect on that database would fail until the transaction is committed. You can now properly connect to the created database.

c-treeDB now Correctly Converts String to CTMONEY

ctdbStringToMoney() incorrectly converted a string to *CTMONEY* when only one decimal digit was supplied. For example: "1.2" should have been converted to the *CTMONEY* value 1.20. Instead, the function converted the value to 1.02. This conversion is now properly handled.

Corrected CTNumber Declarations

The c-treeDB C++ API Declarations for *CTNumber* were found to be in error. The correct declarations are now properly implemented as follows:

```
CTNumber& operator+=(const CTNumber& value)
    {add(*this, value); return *this;}

CTNumber& operator--=(const CTNumber& value)
    {sub(*this, value); return *this;}

CTNumber& operator*=(const CTNumber& value)
    {mul(*this, value); return *this;}

CTNumber& operator/=(const CTNumber& value)
    {div(*this, value); return *this;}
```

Corrected Handling of Empty Variable Length Fields with c-treeDB

While querying records from imported tables with c-treeSQL, it is possible to have records returned containing incorrect data. Fields in the variable length portion of the record returned by c-treeSQL contained wrong content.

A c-tree record is not required to have data in the variable portion, yet the c-treeDB logic had an expectation of such. In the case of a previous record containing data in the variable part and when doing a **ctdbNextRecord()** operation this data was not removed. The problem could then be observed when the next record had no data in the variable length, and the previous record's data appeared in this field.

To resolve this problem, c-treeDB now fills the end portion of the record buffer with nulls ('\0'), after the record is read, effectively clearing the record buffer.

Properly Built c-treeDB Target Keys with Conditional Indexes

If a table has a conditional index, and **ctdbFindRecord()** is called on that index, c-treeDB fails to build a proper target key if the record buffer contents fail the conditional expression. If the record buffer

contents fail the conditional expression, the target key buffer is filled with nuls (`\0`). If this key is then used in a find operation, unexpected records can be returned.

c-treeDB now avoids the conditional expression validation of the record buffer contents when building the key. This results in a properly formed key, and correct results from a query operation.

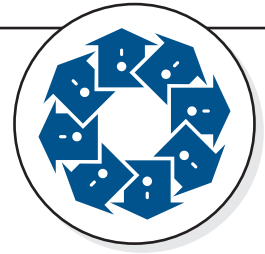
CTDBRET_NOSEGMENT Returned on Alter Table and Create Table

The `ctdbCreateTable()` function has been adjusted to check if the index handles have at least one segment each. If at least one index has no segment, the `ctdbCreateTable()` function return error **CTDBRET_NOSEGMENT** (4095) - index has no segments.

The `ctdbAlterTable()` function was also adjusted to check if any of the index handles have at least one segment. If at least one index handle has no segment, the `ctdbAlterTable()` function returns error **CTDBRET_NOSEGMENT** (4095) - index has no segments. `ctdbAlterTable()` was also changed to mark an index for deletion if all its segments are deleted.

ctdbFindRecord() with CTFIND_EQ mode and Indexes Allows Duplicates

Prior to this change, `ctdbFindRecord()` with a mode of `CTFIND_EQ` on an index allowing duplicate key values always failed with error **INOT_ERR** (101). `ctdbFindRecord()` was modified such that when called with a mode of `CTFIND_EQ` on an index allowing duplicate key values returns the first record in physical order whose key fields in the record buffer match a key value in the index.



c-treeACE

6.1 Serious Issues



FPUTFGET Locking Issues

Two independent issues affect FairCom's multi-user standalone operational model (*FPUTFGET*) which relies on the underlying operating system to ensure file locking integrity. This contrasts with FairCom's server-based models where locking is independently and solely maintained by the c-tree Server. These issues manifest in either of the following scenarios:

1. In environments where all machines accessing the data are using Windows Vista, it is possible for one c-tree Plus *FPUTFGET* application to “hang” when obtaining the record lock already owned by another *FPUTFGET* process. This impacts database files from many vendors and a patch from Microsoft is available.
2. In mixed OS environments (Windows XP with Vista for example) indexes can become out of sync resulting in **ITIM_ERR** (160) errors or even data corruption. This impacts versions of c-tree Plus V7 and V8 prior to V8.27 Build 071005 with *HUGE* files enabled, which is the default in V8.14 and later. This can also occur when using the c-tree Plus ODBC Driver in read/write mode and the *FPUTFGET* model of operation.

Applications using the c-tree Server and c-treeSQL Server are not impacted by these issues.

Microsoft Hotfix for Windows Vista Locking Behaviors

There is a known issue with the Microsoft Windows Vista operating system that may impact your observed locking behavior. This behavior affects the c-tree Plus *FPUTFGET* model regardless of *HUGE* file usage. Please refer to the following Microsoft Knowledgebase entry regarding a supported hotfix.

[Access may stop responding when you open a remote database](http://support.microsoft.com/kb/935366/en-us)
<http://support.microsoft.com/kb/935366/en-us>

This hotfix may not be generally available and may need to be specifically requested for this observed behavior.

c-tree Plus FPUTFGET Locking with HUGE files on Windows Vista

FairCom's multi-user standalone operational model (*FPUTFGET*) relies on the underlying operating system to ensure file locking integrity. This contrasts with FairCom's server-based models where locking is independently and solely maintained by the c-tree Server. The release of Microsoft's Windows Vista operating system has impacted *FPUTFGET* based c-tree Plus applications under the following conditions:

- The application is linked with a c-tree Plus V7 or V8 library in multi-user standalone mode (*FPUTFGET*); and
- The c-tree Plus library has *HUGE* files enabled (c-tree technology for files > 4GB); and

Note: *HUGE* files are enabled by default beginning with c-tree Plus V8.14.

- The Microsoft Windows Vista operating system is used in a mixed OS environment (for example, with Windows XP or another OS.)

Symptoms Observed

When a c-tree Plus multiuser standalone (*FPUTFGET*) application was run under Windows Vista and accessed the same data file as an *FPUTFGET* process from a Windows XP (or other OS) machine, the two processes did not respect each others' record locks. Without proper locking, this can cause **ITEM_ERR** (160) errors if an index key value is involved in an update. Under the worst scenario, other non-key field updates could cause the data record to become out of sync resulting in out of sync, or data "corruption".

Note: c-tree Plus cannot determine when this has happened! Depending upon the application, this symptom may not be recognized for long periods of time. In some cases, many months could pass before an observation is made regarding the out of sync data.

c-tree Plus Modification

When c-tree Plus is initialized in *FPUTFGET* mode, c-tree Plus logic checks if the operating system on which the process is running supports *HUGE* files (files > 4 GB). However, the function that returns the operating system version did not recognize Windows Vista as a valid operating system version, as Windows Vista was not available at the time c-tree Plus Version 8 was released. As a result, c-tree Plus did not consider *HUGE* files an option with the Vista OS.

Versions of c-tree Plus after V8.27 (Build 071005) now recognize Windows Vista as a valid version of Windows and additional checks have been added such that new versions of Windows will be automatically recognized as they are introduced. Rather than checking that only Windows NT/2000/2003/XP support *HUGE* files, the logic now assumes that Win32s, Windows 95 and Windows 98 do **not** support files over 4 GB and all other versions do.

c-tree Plus ODBC

This change also affects the c-tree Plus ODBC Driver used to access *HUGE* files with the mentioned operating systems. This is only for those applications with *HUGE* files enabled and use the c-tree Plus ODBC Driver in read/write mode and the *FPUTFGET* model of operation. The above mentioned modifications are available for the c-tree Plus ODBC Driver.

Testing Windows Vista Locking with c-tree Plus FPUTFGET

It is easy to test your c-tree Plus *FPUTFGET* application for this behavior. The c-tree Plus example program, **ctixmg**, provides an easy-to-use application allowing for a simple controlled test. This example is built when you include samples in your c-tree Plus mtree build.

1. Build **ctixmg** choosing the multiuser standalone (*FPUTFGET*) c-tree Plus model.

2. Start two **ctixmg** instances on two different machines. (Vista to Vista or Vista to XP for example)
3. **B)egin** Locking from the first instance of **ctixmg**.
4. **A)dd** a record from the first instance of **ctixmg** and do not **E)nd** Locking.
5. **B)egin** Locking on the second instance of **ctixmg**.
6. Attempt to **U)pdte** the same record as added in Step 4 from the second instance of **ctixmg**.
7. You should receive a **DLOK_ERR** error (42) from the second instance indicating that the expected and appropriate locking has taken place. If your application hangs (Vista to Vista), or the Update succeeds (Vista to XP), then you may require the Windows Vista hotfix and/or a c-tree Plus update.

FPUTFGET c-tree Library Modifications for Consistent Locking

c-tree Plus uses a maximum value to mark the top of the available locking region. You can manually set the top of the c-tree Plus lock region to ensure consistent locking for c-tree files. Add the following **#defines** at the bottom of your c-tree makefile used to build the *FPUTFGET* c-tree library. You will then need to recompile your application and the c-tree Plus library and relink your application to enable these changes.

```
echo #define ctLOCK_TOPhw      ((ULONG) 0x7fffffff)    >>$(fcTOM)\ctoptn.h
echo #define ctLOCK_TOP      ((ULONG) 0xffffffff)    >>$(fcTOM)\ctoptn.h
```

Keep in mind of what Windows operating systems you will be using with this change. While this will allow for proper locking in the c-tree Plus *HUGE* file range, this will not work with Windows 98, for example, as you will receive a seek error when attempting to go beyond the 4GB file size limit of that OS. For these mixed OS environments, it is best to contact FairCom and update to the latest version of c-tree Plus V8.27.

Proper Reference of Unused File Numbers with CLSFIL

A core dump was experienced when **CLSFIL()** was called with a file number not in use. When *ctFLEXFILE* is defined, and depending on other compile time options (in the reported case, single-user, non-*TRANPROC*) it was possible for a call to **CLSFIL()** to attempt to de-reference a NULL pointer. New logic prevents this illegal reference.

ISRL_ERR (554) with SCHSRL During Rebuild Corrected

When **CREIFIL()** was called with an *ISEG* array that included one or more serial number segments defined with *SCHSRL* for the segment mode, it could not completely determine the segment attributes until the *DODA* was added to the file and could return error **ISRL_ERR** (554). An internal routine is used to help manage and check the serial number segments, including translating the *DODA* reference to an actual byte offset in the record, and making sure all serial number segments are internally consistent. However, it did not properly discriminate between file creation and rebuild cases. Internal rebuild calls are now flagged such that the actual serial number record offset can be properly set.

MACOSX - XCODE Syntax Error

A customer reported problems compiling on MacOS X using XCODE. The compiler error was a previous declaration of *ProcPtr*. *ProcPtr* depends on **#ifndef** *ctPortMACOSX* as well as *ctPortMAC*, and these were not brought into the makefile in the proper order. **mtmake** has been modified to bring in the header files in proper order.

terr(8989) Under FPUTFGET Corrected

Catastrophic c-tree terr (8989) is designed to trap a request to prune an index tree under *FPUTFGET*. *FPUTFGET* does not support this action because of node lock deadlock problems. The node cleanup routine has been modified to handle this situation when *FPUTFGET* is defined.

Calling GetSymbolicNames() Now Returns Correct File Name

When **GetSymbolicNames()** was called with a mode of *FIRST_ITEM* or *NEXT_ITEM*, the returned *APP_NAME_LIST* file name sometimes unexpectedly includes a vertical bar (|) character. For example, the file name *mydatafile.txt* might be returned as *myd|tafile.txt*. This issue was corrected by properly handling a previously uninitialized buffer.

Complete Update of File Sync List from Notification

The c-tree notification module may cause invalid memory reference exceptions (access violations or segmentation faults) when processing notifications for variable length records. The notification module adds an entry to its file sync list for each file for which it processes notifications. The file list entries contain information about the data file such as the data file number, record buffer size, and record buffer pointer.

When processing notifications for a file, the notification module reads the file sync list entry for the specified file into a local file sync element structure. If the newly-allocated record buffer's address differs from the original record buffer's address, the file sync list entry's record buffer pointer is no longer valid and a subsequent attempt to reference the memory produces an exception. The logic to maintain these lists has been adjusted to accommodate record buffers of different lengths.

EQLKEY Now Checks for Invalid keyno

The *CT_TIMES* field type is inherently aligned the same as a double float. However, a file receives an alignment attribute based on the alignment characteristics at file creation. If the file is moved to another platform with a different alignment, this is now accounted for.

Unused Debugging Code Removed in Hugefile Support

A customer reported a core dump in the low level routines **ctdshbuf()/ctgetbuf81()**. **ctdshbuf()** maps a file number and file position to a data cache hash bin, and the routine had debugging code that checked if the low order four bytes of a file position are 0xffffffff. This can not happen for a non-huge file, but it could happen for a huge file as it is nearing a 4 GB boundary. The debugging code is no longer appropriate for huge files and has been removed.

ctLOCLIB with ctCLIENT Redirection Solved

It was found through internal testing that calling **ALCRNG()** with a *ctLOCLIB/ctCLIENT* combination could result in a core dump when using the local connection. This has been resolved.

LOCLIB Memory Overwrite Fixed

A memory overwrite clobbered the socket descriptor value used for the client side of the *LOCLIB* connection. Various symptoms could occur, including error **ARQS_ERR** (127) on an **ADVREC()** call. This occurred because the memory allocated for the c-tree global variables for the client connection was too small. **REGCTREE()** allocated the global variable memory using the size of the local-side

c-tree global variable structure rather than the size of the c-tree client-side global variable structure, which happened to be larger than the local-side version of the c-tree global variable structure. As a result, when accessing fields near the end of the c-tree global variable structure, the c-tree client-side code read and wrote past the end of the allocated c-tree global variable buffer. Logic was added to ensure a proper memory allocation is performed for the c-tree global variables.

ctSysQueueRead Hang on Empty Queue Corrected

A call to **ctSysQueueRead()** with a zero timeout parameter did not return if the queue was empty, and the server was in a fairly tight loop consuming excessive CPU cycles. **ctSysQueueRead()** is designed to be interruptible if it is called with a non-zero timeout. By interruptible we mean that if the user is terminated, say by an administrative process, the user thread will promptly detect it has been terminated and the wait for the queue to have an entry will be interrupted. However, the code did not properly handle the zero timeout case; it never attempted to block on the queue control structure when the queue was empty, and kept looping to check the queue status.

The zero timeout case (as well as the non-zero timeout case) must attempt to block on the queue control structure. If the queue is empty and the timeout is zero, the blocking call will immediately return with error **NTIM_ERR** (156) which will be returned by the **ctSysQueueRead()** call.

Access Violation on c-tree Superfile Member Create or Open

An unhandled exception could occur when creating or opening a c-tree Superfile member. If the c-tree code was compiled with read-only string pooling enabled and a string constant Superfile member name is passed to a file create or open function, an access violation occurs when c-tree attempts to overwrite the exclamation point in the Superfile member name with a nul byte. The access violation occurs because the process is trying to modify memory that is marked read-only. We now copy the Superfile member name to a buffer allocated on the stack and modify the copy of the file name on the stack.

Extended Create Block Function Corrected

If a c-tree client passed a NULL extended create block pointer to an 8-byte extended file create function, **CreateISAMXtd()**, an unhandled exception occurred in the c-tree code. A check is now made to avoid dereferencing the NULL pointer.

Correct File Open with a Unicode Key Segment on an Index Member

It was possible to experience a core dump in either server or stand-alone code when opening a file with a Unicode key segment on an index member. In the case of Unicode information pertaining to an index file, the code to extract information from the resources was executed prematurely. This worked whenever the Unicode information pertained to the index as a whole or the host index. If the information pertained to one of the subsequent index file members, c-tree attempted to process the member information before the member had been completely initialized. This resulted in an unexpected NULL pointer condition. This logic has been adjusted to extract the extended key segment information separately

Correct Fileword Corruption in OPNRFILX

Error **SPWD_ERR** (457) was reported on a call to **OPNRFILX()**. It was found the *fileword* parameter was stored in the communications buffer area and was overwritten by **OPNRFILX()** results. This has been resolved by creating a local copy of the *fileword* string.

6.2 Other Issues

Current ISAM Positions after Batch Insertions

The batch insertion feature restored the current ISAM position after the insertions completed. However, if the current ISAM position was for a deleted record (which is a legal situation), the logic in the batch insertion did not deal with this case, and a **ITIM_ERR** (160) was returned.

Now, when the batch insertion behavior is consistent with the batch delete behavior: the last record operated on in the batch becomes the current ISAM position. This modification both changes the behavior and removes the **ITIM_ERR** when a delete determines the current ISAM just before a batch insertion.

Records on LOW_HIGH Platforms

A batch retrieval using the *BAT_PHYS* mode (return records in physical order) for a fixed length data file mistakenly returned deleted records along with active records on *LOW_HIGH* platforms.

The *BAT_PHYS* record-scan reads the first two bytes of each fixed length record into a two-byte integer to test for a resource or deleted record. The byte ordering affects how to test for a leading 0xff byte (a deleted record). The detection logic did not distinguish between a *LOW_HIGH* and *HIGH_LOW* platform thus the test failed on *LOW_HIGH* platforms. The server now accounts for the byte order of the platform to determine the proper status of the records.

Record Offsets of Zero for Physical Order Batch Reads Supported

It was found that a batch read in physical order, with a zero record offset, failed with error **INOT_ERR** (101). c-tree now considers a batch read in physical order with a zero record offset to indicate the physical read is to start with the first record in physical order in the data file. In the case of c-tree Superfile members, the physical scan starts at the position of the last record added to that Superfile member.

Support System Queue Functions in LOCLIB mode

System queue functions called in *LOCLIB* mode previously returned error **NSUP_ERR** (454). The expected behavior is that a *LOCLIB* client connection supports the system queue functions and a *LOCLIB* local connection does not support the system queue functions and this feature is now turned on for the *LOCLIB* mode.

Correct Return Values of System Queue Functions

The functions **ctSysQueueOpen()**, **ctSysQueueMlen()**, and **ctSysQueueCount()** are documented as returning error codes as negative values, however, were found to return error codes as positive values. The logic was adjusted such that the proper negative values are now returned for these functions.

VDLK_ERR During RETVREC Call in Single User Mode

Error **VDLK_ERR** (146) was reported during a **RETVREC()** call.

In single user mode, an attempt to coalesce a variable length record being deleted along with the space immediately trailing the record assumed that if the trailing record has a delete record mark, it can be coalesced. Checking the index in this case was skipped as it was assumed that the lack of multi-user interference made the check superfluous. However, in this single user application, a **NEWVREC()** call caused space to be made available, and at the time of the delete, the “new” record space had still not been used and the record mark still indicated deleted space. But, the space management index had already had the entry for the new record removed in the **NEWVREC()**. If this “new” (unused) record happens to immediately follow the space being deleted, the **VDLK_ERR** error occurs when the trailing space is not found in the space management index. The single user model now checks for the trailing deleted space’s entry in the space management index before trying to coalesce

Proper Deny Open of a Mirrored Pair of Segmented Files

It was found that if a user, outside of c-tree, copied a segmented file (host) into another file, and then attempted to open the file using the “primary|mirror” naming convention, the open would succeed. c-tree should not (and does not) permit the creation of a segmented file host that is mirrored.

Transaction Control of Conditional Index Expressions Corrected

A problem was found in which transaction control is maintained over a disk image of a resource and that resource, say at file open, gets mapped into a memory construct for the purposes of actual execution. The code for the conditional index expressions essentially assumed that the creation of the resource was not part of an arbitrarily large transaction, but rather a single update that succeeded or failed. So only simple, local control of the associated memory construct was maintained.

For example, the following pseudo-code would lead to an unexpected result:

```
TRANBEG
SAVEPOINT #1
UPDCIDX(key1,expr1)
RESTORE to savepoint #1
TRANEND
```

```
TRANBEG
ADDRUC
TRANEND
```

The **ADDRUC()** in the second transaction would still be affected by the undone **UPDCIDX()** because while the **RESTORE()** removes the conditional expression resource, it did not undo the memory construct that the file header uses to implement the conditional index. An improved method of handling this situation has been enabled.

FreeRange (FRERNG) Header File Fix

c-tree Plus V8.14 implemented a new function, **FreeRange()** (Short name **FRERNG()**). When a user attempted to use the long name, a link error was reported. The macro to map the long name to the short name had inadvertently been skipped. This macro has been added to allow for proper linking.

Assignment of updateIFIL Now Preserves Previous IFIL Setting

During a support incident it was noticed that an update *IFIL* could overwrite previous *IFIL* settings. This problem has been corrected.

Old fileIDs Properly Properly Removed During Rebuild

During a rebuild, the 12-byte unique *fileIDs* maintained by the c-tree Server are reset for the data and index files so that automatic recovery does not confuse the pre and post instances of the files. It was found that an internal routine did not remove the old *fileID* from the index which maintains the list of unique IDs for the currently active files. This combined with copying a file could lead to a conflict with an obsolete *fileID* on an open of the copy of the file. To avoid this behavior we remove the old *fileID* from *NAMINDEX* before reconstructing and add the new *fileID* to *NAMINDEX*.

Corrected Serial Segment Test

When using the commercial V8.14 release of c-tree Plus, error **NSUP_ERR** (454) resulted when using a serial segment modified by *DSCSEG* (or any other segment modifier) where it had worked with a previous c-tree version (V6).

ctsrseg() checks for an appropriate serial segment and sets the length and position (in the record) of the serial segment. The test for a *SRLSEG* or *SCHSRL* did not first mask-out the segment modifiers. Hence a statement of the form

```
if (segmod == SRLSEG) ...
```

would fail, even if the segment mode was a *SRLSEG*, whenever the segment mode also included one or more segment modifiers. The modifier bits are now masked before checking on the type of serial segment.

BLKIREC Properly Finds all Matching Records

When **BLKIREC()** is called with a target key containing a key segment that is sensitive to multiple key transformation operations and automatic ISAM key transformation is enabled, **BLKIREC()** did not find a matching record even though it existed. By “sensitive to multiple key transformation operations”, we mean repeated calls to transform the key segment cause the key segment value to change with each key transformation operation. An *INTSEG* segment on a *LOW_HIGH* system is an example of this. By contrast, a *UREGSEG* segment is not sensitive to multiple key transformation operations.

When automatic ISAM key transformation is enabled, **BLKIREC()** calls **TFRMKEY()** to transform the target key and then calls an ISAM level record retrieval function which also calls **TFRMKEY()**. The solution was to remove the key transformation call from **BLKIREC()**.

LTEREC Rproperly Returns Record with Key Value Equal to Target

When a client called **LTEREC()** to retrieve a record whose key value is less than or equal to the target key value, and the index allows duplicates and automatic ISAM target key transformation was enabled, a record whose key value is less than the target key value was returned even when a record exists with a key value equal to the specified target key value.

The automatic transform key operation sets the target key's suffix (record offset) bytes to zero. This causes the search to miss key values that are equal to the target key value.

For **LTEREC()** operations, the internal key transformation operation now sets the target key's suffix bytes to the maximum record offset value

Note: This modification does not change the behavior of **TFRMKEY()**. This means when a c-tree application calls **TFRMKEY()** manually (such as when calling **LTEKEY()** or when calling **LTEREC()** with automatic ISAM key transformation disabled), the application must fill the record offset with 0xff bytes after the **TFRMKEY()** call.

vtclose no Longer Fails in FPUTFGET Mode

When an application which was linked with a multi-user standalone c-tree library opened more than the maximum number of virtual files (as determined by `#define MAXVFIL`, which defaults to 500) on a Unix system, c-tree logs the following error message to `CTSTATUS.FCS`:

```
vtclose failure:  LOC 1:  TOT 501:  CHN 0  NOV 0  LOK 500  NFD 0  IOS 0
```

A previous modification enabled exclusive file open support in c-tree multi-user standalone mode on Unix. When exclusive file open support for Unix is enabled, c-tree acquires a lock on each open c-tree file in order to support exclusive and shared file opens. But, acquiring the lock on the file increments the file's lock count, and files with non-zero lock counts cannot be virtually closed because the locks would be released when c-tree closes the file. As a result, when the number of open virtual files reaches the open virtual file limit and an attempt is made to open another virtual file, c-tree attempts to find a virtual file to close but does not find an available virtual file to close and logs the `vtclose()` error message to `CTSTATUS.FCS`.

Multi-user standalone mode on Unix now supports either exclusive file opens or virtual files but not both in a single compilation of the c-tree library.

c-tree Status Log Write Failure Causes Process to Exit

When a situation occurs that causes the c-tree standalone library to write a message to the c-tree status log, `CTSTATUS.FCS`, and the status log cannot be opened, the process terminates.

The c-tree status log write function, `ctrcvlog()`, considers a failure to open the c-tree status log to be a fatal error and when this happens, it exits the process. However, exiting the process is not desirable from the point of view of the application developer and end user. Also, it can be common for the status log to fail to be opened, in particular when opening many files on Unix systems causes the process to reach the system limit on file descriptors per process.

Rather than exiting the process, the status log function now returns an error if the status log cannot be opened. If the c-tree library is compiled with `#define RB_OUTPUT`, then when the status log cannot be opened the message that was to be written to the status log is written to the standard error stream. To restore the previous behavior (the process exits when the c-tree status log cannot be opened), uncomment `#define ctPREV_STATUSLOG_FAIL_EXIT` in `ctopt2.h`.

Support Range Functions when Huge File Support is Disabled

When c-tree is compiled without huge file support, the c-tree range functions return error `NSUP_ERR` (454). This is because the non-huge version of the range functions are coded to simply return with error `NSUP_ERR`. This modification applies the changes needed to support range functions when c-tree's huge file support is disabled.

Compiling the Standalone Model Without c-tree Superfile Support

c-tree failed to compile in standalone mode without c-tree Superfile support due to a mismatched curly brace and an undeclared identifier. This modification corrects these errors and warnings about unused and uninitialized variables in this configuration.

Correct Open of File with Unicode Key Segment

Attempting to open a file with a Unicode key segment failed with error `ICUV_ERR` (750).

A Unicode extended key segment definition specifies the version of ICU used when the index was created. If the ICU version used to open the file differs from the ICU version used to create the index,

c-tree fails the open attempt with error **ICUV_ERR** , different ICU version, rebuild index). However, an internal function did not properly compare the file's ICU version and the current ICU version. The function compared the addresses of the array used to store the two versions rather than comparing the values. This comparison has been corrected.

Transaction Commit no Longer Fails with **SEEK_ERR (35)**

When a non-transaction-controlled *WRITETHRU* file is created and closed before a transaction commit in standalone mode, the transaction commit failed with error **SEEK_ERR (35)**. The transaction control logic has been adjusted to skip files not open thus avoiding this condition.

Improved Sensitivity of **ESTKEY** Routine

It was reported **ESTKEY()** sensitivity was affected by the size of an index node. As the node got larger, it seemed **ESTKEY()** results were less accurate. **ESTKEY()** approximates the number of index entries between two key values. It performs a small number of index tree walks to create the estimate. The estimation algorithm was adjusted to give a greater sensitivity range, with a relatively small number of additional tree walks.

ISAM Context Data Properly Updated During Index Create or Delete

It was found a record update following a record read in another context fails with the error **KDUP_ERR (2)** after adding an index to a file for which an ISAM context had been established for all indexes. The ISAM context information was not updated to track the current key buffer for the new index. A flag has been added to indicate which ISAM contexts have been established on all indexes for a data file. For such contexts, when an index is created, the context information is updated to include the new index.

Correct 8-byte Serial Number Usage

It was found that the record buffer for a newly-added record for a data file that has an extended header with an 8-byte serial number (*SRLSEG*) key segment showed an incorrect serial number. Only four of the serial number's bytes were set in the record image returned to the client. A macro change to properly check the extended header corrects this problem.

It was also found on *LOW_HIGH* systems, an 8-byte serial number did not have its upper four bytes set. It was determined that an improper use of a pointer address resulted in this failure, and this has also been corrected.

REDIREC() no Longer Fails in Multi-User Multi-Threaded Mode

It was reported the function **REDIREC()** returned error **ZREC_ERR (29)**, specified record offset is zero, when called in multi-user multi-threaded mode even though the application passes a non-zero record offset to **REDIREC()**.

When the Visual C 6 compiler compiles c-tree code with global optimizations enabled (using the */O2* compiler option, which turns on the */Og* option), the compiler generates incorrect instructions for an internal **ctree()** function and caused a zero record offset to be passed. A disassembly showed the record offset passed to **ctree()** was stored in the *edx* register but that *edx* was zeroed out when evaluating the **switch()** statement determining which function pointer is to be used.

Disabling global optimizations when compiling the **ctree()** function when using the Microsoft Visual C V5 or V6 compiler resolves this problem.

Rebuild Now Correctly Purges Records with Duplicate Keys when c-tree is Initialized with USERPRF_PTHTMP

It was found a call to rebuild indexes and purge duplicate key values in c-tree standalone mode failed with error **KDUP_ERR** (2) if the *USERPRF_PTHTMP* user profile mode was specified when initializing c-tree. An uninitialized buffer was passed to a temporary file name functions resulting in a failed file creation. The inappropriate buffer is now cleared before use.

Stable Index Sizes after Automatic Recovery

Index nodes and data records are treated very differently with regard to transaction log entries. Data record images are stored directly in the log, but index node images are not stored in the log. As a result, the treatment of deleted nodes which are placed on a node delete stack for reuse is very different from the delete stack of fixed length data records which is maintained during automatic recovery.

Previously with c-tree, if an index was recovered without the *LOGIDX* option, the delete stack was lost during recovery. Further, the updates to the header of an index as nodes were pushed onto the top of the delete stack were not written to disk until the index file was closed.

The new default for the single user *TRANPROC* model introduces the following new behaviors:

1. The index file header is written to disk when the top of the node stack changes.
2. During recovery, an attempt is made to maintain the delete stack whether or not *LOGIDX* is enabled.
3. If a defect is found in the delete stack during recovery, the good portion (if any) is kept.

ctCAMO support for c-tree Client Compiles

To use file encryption (basic or advanced encryption), a client must be compiled with encryption enabled. By default, a c-tree client library built with *mtmake* previously did not have this support. A call to **ctSETENCRYPT()** thus failed with **NSUP_ERR** (454, feature not supported). This encryption support is now enabled by default for all client builds.

Rebuild Callback with LOCLIB Mode

The call to **SETCBRBL()** failed in the c-tree *LOCLIB* model. Support has been added to enable rebuild callback support with this model.

Correct NTIM_ERR Returned by SYSMON() when a Time-out Occurs

It was reported that **SYSMON()** returned a system error code instead of **NTIM_ERR** (156) when it timed out. The server-side function used to perform an interruptible wait now returns **NTIM_ERR** instead of the system return code in this situation.

Correct Byte Ordering of nul Padded CT_STRINGs with the c-tree Plus ODBC Driver in Heterogeneous Environments

In a heterogeneous environment, a c-tree client reverses binary fields in the record buffer when reading data records. A *CT_STRING* field is considered to end when the first null byte is encountered. If an application pads a *CT_STRING* field with multiple nul (0x0) bytes, the c-tree client's scanning logic stops scanning the field at the first nul byte found and considers the next field to begin with the second nul byte. This can cause the client to reverse the wrong bytes in the data record buffer.

The c-tree Plus ODBC Driver has been enhanced to scan a c-tree data file's field definitions when the client reads the *DODA* from the data file and to change all *CT_STRING* fields having a non-zero field length in the fixed length portion of the record buffer to *CT_FSTRING* fields. This avoids having to scan through the strings, and allows the c-tree Plus ODBC Driver to use a pre-defined length for the string.

Corrected Lock Dump Output

Lock dump output displayed a record offset of 0000-ffffff92415991x that should instead have read 0000-92415991x. This problem was observed with a 64-bit c-tree Server running on the Sun Solaris operating system.

The variable passed to `fprintf()` is a 4-byte signed integer and the format string uses the `%lx` specifier. The `%lx` format specifier for a 64-bit process appears to cause the input value to be treated as a 64-bit integer. As the variable is a signed integer, the value has its sign extended to the high word of the 64-bit integer.

This record offset value is now cast as an unsigned long such that the sign is not extended into the high word, providing the expected output.

Properly Initialized `CMPIFIL()` Callback Variables

When a c-tree client calls `SETCBRBL()` to set a rebuild callback function and then calls `CMPIFIL()` to compact a c-tree data file, the c-tree Server could terminate with an unhandled exception.

For a `CMPIFIL()` operation, the c-tree Server did not initialize the client communication variables used by the server-side rebuild callback function. c-tree Server logic was modified to initialize the client communication variables used by the server-side rebuild callback function when a `CMPIFIL()` call is made with a rebuild callback function set by the client. The c-tree client logic was also modified to set the message output length and to call the client-side rebuild callback function in this situation.

Improved Handling of Header Values Prevents `KDUP_ERR` with `SRLSEG` Keys

An unexpected `KDUP_ERR` (2, duplicate key found) was encountered while using a `SRLSEG` value as the key. `PUTHDR()` behavior has been changed such that on commit, the header value is not reset to the new value. Abort behavior remains the same. This change also affects `numrec1`, `numrec2`, `phyrec1`, and `phyrec2` header values.

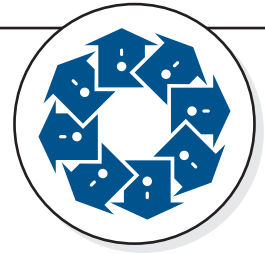
Note: For backward compatibility, the original behavior can be restored by adding `COMPATIBILITY PUTHDR_COMMIT` to the server configuration file.

The problem can be demonstrated in the following pseudocode in which we assume a unique key based on a `SRLSEG`:

```
TRANBEG
CREIFIL
PUTHDR -> set serial number to 3
ADDREC -> serial number incremented to 4
TRANEND -> changes serial number back to 3
TRANBEG
ADDREC -> KDUP_ERR on key with SRLSEG
```

The `SRLSEG` values (`sernum1`, `sernum2`) are stored in the c-tree Plus file header. Header values are handled slightly 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() logic uses a traditional transactional approach whereby old values are restored upon abort and the values at the time of the **PUTHDR()** call are stored upon commit. In the case described above this results in a reset of the value. It is this resetting of values that sets the conditions for the **KDUP_ERR** symptom.



c-treeACE Utilities

7.1 Successful Rebuild of Segmented Files With no File ID

The c-tree *IFIL*-based Rebuild Utility, **ctrbldif**, linked with a c-tree single-user library with transaction support terminates with an unhandled exception when rebuilding the indexes of an automatically segmented (*ctSEGAUTO*) file that did not have a file ID set. A file was not assigned a file ID if it was created using a single-user library without transaction support. A pointer assignment has been adjusted to correct this exception.

7.2 Correct Restore of a Segmented Dynamic Dump File

The dump restore utility, **ctrdmp**, terminated with an unhandled exception when processing the second segment of a segmented dynamic dump stream file. An uninitialized variable was found and could occasionally result in a NULL pointer reference. This variable is now properly assigned at the entrance to the function.

7.3 Avoid Unnecessary XCRE_ERR Error During Rebuild with Missing Indices

During a rebuild of a c-tree HUGE file, an **XCRE_ERR** (669) could be observed when the indices did not exist, or the rebuild utility attempted to create a new index file. An internal inconsistency was created with regard to extended file modes. A *HUGEFIL* attribute was detected in the data file, however, the default create block was created without extended headers. **ctrbldif** now attempts to use a default *HUGEFIL* extended create block (*XCREBLK*) to re-create the index avoiding the **XCRE_ERR**.

7.4 c-tree Restore Utility Opens all ctTRNLOG files Restored from Dump

When restoring a large number of files from a dynamic dump backup with **ctrdmp**, it was found that the recovery phase of the dump restore could fail with error **FNUM_ERR** (22, file number out of range). An increased number of c-tree files is now used to avoid this situation.

Should you still encounter this error with very large number of restored files, consider the **!#FCB** option in your **ctrdmp** script file to increase this value.

7.5 IFIL-Based Rebuild Utility Correct Functions with Both -purge and -updifil

When the c-tree *IFIL*-based Rebuild Utility, **ctrbldif**, was run with both the *-purge* and *-updifil* options specified, the utility behaved as though neither option was specified. The utility has been modified to use the correct behavior.

7.6 Flush Standard Output Before ctclntrn and cthghtrn Utilities Exit

When running the c-tree Transaction Highwater Mark Utilities, **ctclntrn** and **cthghtrn**, these utilities appeared to not properly redirect output to a file. The standard output should be flushed before the process exits in order to ensure that all output is written to the file. System flush calls have been added to the **ctclntrn** and **cthghtrn** utility functions before exiting.

7.7 dbload Memory Overwrite Corrected

The c-treeSQL data load utility, **dbload**, was found to crash in a common usage. A discrepancy in array values between the client and the server was discovered, resulting in the client utility to overwrite memory. This discrepancy has been resolved, preventing further utility failures

7.8 c-treeSQL dbdump Crash Corrected

It was found that a simple c-treeSQL Dump Utility, **dbdump**, script could cause **dbdump** to crash. The code checking for syntax errors overwrote memory when building the error message. We now allocate enough space for the strings involved.

7.9 Correct Monitoring of a Large Number of Clients with the c-tree Statistics Utility.

The c-tree Statistics Utility, **ctstat**, terminates with a segmentation fault when used to monitor a large number of connected clients or files.

ctstat declares arrays of user and file statistics structures. When the number of connected clients exceeded the array size, **ctstat** inadvertently wrote past the end of the array, terminating with a segmentation fault.

A check was added to ensure **ctstat** does not write past the end of these arrays. If the number of entries to display exceeds the array size, **ctstat** now displays the following message, where <displayed> is the number of entries displayed by **ctstat** and <total> is the total number of entries **ctstat** read from the c-tree Server:

```
# Note: Limited output to <displayed> of <total> entries.
```

7.10 ctstat Utility Returns Exit Code 101

If the c-tree Server Statistics Utility, **ctstat**, is run for a fixed number of iterations, an error code of 101 is generated when the utility exited. This behavior was confusing, as the utility actually executed without errors.

The **ctstat** utility uses error **INOT_ERR** (101, record not found) to break out of the loop in which it reads **SNAPSHOT** data from the c-tree Server. This error code is returned to the main function and becomes the process exit code.

The new behavior is now if the error code is **INOT_ERR**, we set the error code to zero (0, **NO_ERROR**).

This modification also adds a new check for a negative or zero interval command-line argument. If a negative or zero interval is specified, it is automatically converted to an interval of 1 second. This prevents a zero interval from being used, which would cause **ctstat** to repeatedly read **SNAPSHOT** statistics without pausing, and put an undesirable load on the c-tree Server.

7.11 sa_admin now adds New User Accounts to Specified Group

When the c-tree command line Server Admin Utility, **sa_admin**, was used to create a new user account and a group name is specified for the user account, the user account is created but is not associated with the specified group. **sa_admin** failed to increment a counter indicating the number of groups associated with the new user account. This counter is now incremented such that **sa_admin** adds the user to the specified group.

7.12 ctstop Utility now Exits with a Non-Zero Return Code

When the c-tree Server Stop Utility, **ctstop**, fails to connect to the c-tree Server, it now displays the following message and exits:

```
"Could not logon to server. Error #<err>."
```

Previously in this situation, **ctstop** continued execution and prompted the user:

```
"Do you still wish to terminate the server?"
```

7.13 Dump Index Utility Support for Huge Files

The c-tree Dump Index Utility, **ctdmpidx**, now displays 64-bit values read from a huge file's header and from nodes in huge index files. Previously, **ctdmpidx** only displayed the low-order word of the 64-bit values.

7.14 c-tree Configuration Settings Utility no Longer Requires Write Permissions

The c-tree Configuration Settings Utility, **ctcfgset**, no longer requires write permissions on the configuration file when creating an encrypted c-tree Server settings file. The utility has been adjusted to only require read permissions on the plain text c-tree Server configuration file.

7.15 Correct Mirrored File Handling with the Superfile Prepass Utility

While mirrored Superfiles could be created and updated successfully, calling the c-tree Plus Superfile Prepass Utility, **CTSBLDX()**, with a mirrored Superfile file name syntax such as *file1|file1mirror* failed with **FNOP_ERR** (12), "Could not open file". The complete filename was passed, including the mirrored portion, and was improperly handled by the Superfile Prepass Utility.

The Superfile Prepass Utility was modified by including *ctMIRROR_SKP* in the file mode when the Superfile and its members are opened. This permits the full mirrored name (*file1|file1mirror*) to be

passed to the utility, and operations are only performed on the primary Superfile. When the utility completes its operation, the primary Superfile is ready for its ISAM constituents to be rebuilt, but the mirror is no longer in sync with the primary. Once the Superfile members have been rebuilt, the primary Superfile must be copied into the mirror. (For stand-alone ISAM files that are mirrored the copy operation is automatic upon completion of the rebuild, however, this is not the case for mirrored Superfiles and their members.)

7.16 Correct Usage Output for IFIL-Based Rebuild Utility

The command-line usage output for the c-tree IFIL-based Rebuild Utility, **ctrbldif**, indicated an incorrect number of optional arguments. The output has been corrected to the following:

Client/Server

In client/server mode, this utility accepts one required and five optional arguments:

```
# ctrbldif DataFileName [-purge] [-updifil] [<UserId> [<UserPassword> [<ServerName>]]]
```

where *-purge* indicates that duplicate records should be purged, and *-updifil* that *IFIL* resource in data file should be updated.

Standalone

In standalone mode, this utility accepts one required and three optional arguments:

```
# ctrbldif DataFileName [-purge] [-updifil] [-<sectors>]
```

where *-purge* indicates that duplicate records should be purged, and *-updifil* indicates that the *IFIL* resource in the data file should be updated, and *<-sectors>* is the sector size to use.

Index

- 1**
 - 101 INOT_ERR 44, 54
 - 127 ARQS_ERR 42
 - 146 VDLK_ERR 44
 - 156 NTIM_ERR 43, 49
 - 160 (ITIM_ERR) 44
 - 17 DCRAT_ERR 6
 - 17084 - Maximum Users Exceeded 13
- 2**
 - 2 KDUP_ERR 48, 49
 - 20008 - Inconsistent types 6
 - 20008 (SQL inconsistent types) 13
 - 20029 - Index referenced xx not found 10
 - 20125 - Request for cursor failed 3
 - 20125 (SQL no cursors available) 13
 - 20127 (SQL_ERR_BAD_PARAM) 10
 - 20212 - Error in network deamon 4
 - 20232 - Invalid number string 6
 - 22 FNUM_ERR 53
 - 29 ZREC_ERR 48
- 3**
 - 35 SEEK_ERR 48
- 4**
 - 4025 CTDBRET_INTERNAL 7
 - 454 NSUP_ERR 44, 49
 - 457 SPWD_ERR 43
- 5**
 - 519 IEOF_ERR 26
 - 554 ISRL_ERR 41
- 6**
 - 669 XCRE_ERR 53
- 7**
 - 7054 3
 - 707 USEG_ERR 25
 - 750 ICUV_ERR 47
- A**
 - addBatch (JDBC) 12
 - APP_NAME_LIST 42
 - ARQS_ERR (127) 42
 - automatic recovery
 - index sizes 49
 - L64 error corrected 6
- B**
 - BAT_PHYS 44
 - batches
 - LOW_HIGH platforms 44
- physical order 44
- BLKIREC
 - see BlockingISAMRead 46
- BlockingISAMRead 46
- C**
 - Configurations Settings utility 55
 - CONSOLE NO_MESSAGEBOX 24
 - CreateIFile 41
 - CREIFIL
 - see CreateIFile 41
 - CT_USER
 - shared library support 17
 - ctCAMO 49
 - ctcfgset 55
 - ctclntrn 54
 - CTDBRET_INTERNAL (4025) 7
 - ctdmpidx 55
 - cthghtrn 54
 - CTNumber class conflicts 29
 - ctrldif 53, 54, 56
 - ctrdmp 53
 - CTRecord.GetFieldAsFloat 29
 - c-tree Plus for .NET
 - CTNumber class conflicts 29
 - dispose method 29
 - functions
 - CTRecord.GetFieldAsFloat 29
 - Dispose 29
 - c-treeSQL
 - SYNONYMS 6
 - ctSETENCRYPT 49
 - ctstat 54
 - CTSTATUS MASK_WRITE_ERR 20
 - ctstop 55
 - ctSysQueueCount 44
 - ctSysQueueMlen 44
 - ctSysQueueOpen 44
 - ctSysQueueRead 43
 - zero timeout handling 25
- D**
 - dbdump 54
 - dbload, data load utility 54
 - DCRAT_ERR (17) 6
 - DIAGNOSTICS LOWL_FILE_IO 18
 - DIAGNOSTICS WRITE_ERR_DUMP 20
 - Dispose 29
 - Dump Index utility 55
 - dump utility (c-treeSQL) 54
 - duplicate key errors
 - see KDUP_ERR (2) 49
- E**
 - encryption 49
 - errors
 - ARQS_ERR (127) 42

CTDBRET_INTERNAL (4025).....	7	ISRL_ERR (554)	41
c-treeSQL		ITIM_ERR (160)	44
17084 - Maximum Users Exceeded.....	13	J	
20008	13	JDBC	
20008 - Inconsistent types	6	addBatch	12
20029 - Index referenced xx not found	10	K	
20125	13	KDUP_ERR (2)	48
20125 - Request for cursor failed.....	3	L	
-20127 bad parameter	10	L64 error.....	6
20212 - Error in network daemon	4	LEFT OUTER JOIN.....	8
20232 - Invalid number string	6	Linux	
DCRAT_ERR (17).....	6	CT_USER shared library support.....	17
FNUM_ERR (22).....	53	LOCAL_DIRECTORY	6
ICUV_ERR (750).....	47	LOCLIB model	
IEOF_ERR (519).....	26	memory overwrite fixed	42
INOT_ERR (101).....	44, 54	LTEREC	
ISRL_ERR (554)	41	see GetLERecord.....	46
ITIM_ERR (160).....	44	N	
KDUP_ERR.....	48	NLM path references.....	20
KDUP_ERR (2)	49	notification	
L64	6	invalid memory exception fix	42
NSUP_ERR (454)	44, 49	NSUP_ERR (454)	44, 49
NTIM_ERR (156)	43, 49	NTIM_ERR (156)	43, 49
SEEK_ERR (35).....	48	O	
SPWD_ERR (457)	43	ODBC	
terr (7054)	3	SQLDriverConnect.....	12
terr (8989)	42	SQLExecDirect.....	12
USEG_ERR (707)	25	SQLGetData.....	12
VDLK_ERR (146).....	44	SQLGetDiagField	11
XCRE_ERR (669)	53	SQLStatistics.....	11
ZREC_ERR (29)	48	R	
EstimateKeySpan		ReadIsamData	
sensitivity.....	48	FPUTFGET failure	48
ESTKEY		rebuild	
see also EstimateKeySpan	48	LOCLIB support	49
F		Rebuild IFIL utility.....	53, 54, 56
field types		REDIREC	
CT_TIMES	42	see ReadIsamData	48
FNUM_ERR (22).....	53	Restore utility.....	53
FreeRange	45	S	
FRERNG		sa_admin.....	55
see FreeRange	45	scalar functions	
G		in WHERE clause.....	7
GetLERecord	46	SCHSRL segment mode	
GETNAM		rebuilding.....	41
see GetSymbolicNames	42	SCO	
GetSymbolicNames	42	daemon accept error corrected.....	4
I		Security Administration Utility.....	55
ICUV_ERR (750).....	47	SEEK_ERR (35).....	48
IEOF_ERR (519).....	26	serial segment	
indexes		incorrect 8-byte numbers	48
rebuilding.....	53, 54, 56		
sizes after automatic recovery	49		
INOT_ERR (101).....	44, 54		

server configuration keywords		
CONSOLE NO_MESSAGEBOX.....	24	
CTSTATUS MASK_WRITE_ERR.....	20	
DIAGNOSTICS LOWL_FILE_IO.....	18	
DIAGNOSTICS WRITE_ERR_DUMP	20	
LOCAL_DIRECTORY	6	
server shutdown timeout.....	23	
server write errors	20	
SETCDBRBL	49	
SIGNAL_READY long filename syntax	26	
SNAPSHOT		
negative statistics output.....	24	
Solaris		
direct I/O and O_SYNC.....	25	
SPWD_ERR (457)	43	
SQL keywords		
SYNONYM	6	
SQL Panic		
20125 Request for cursor failed fixed	3	
parameters in a predicate clause fixed	4	
SQL_ERR_BAD_PARAM (20127).....	10	
sql_server.log.....	10	
SQLDriverConnect.....	12	
SQLExecDirect.....	12	
SQLGetData.....	12	
SQLGetDiagField	11	
SQLStatistics.....	11	
SRLSEG	48	
Statistics utility.....	54	
Stop Server utility	55	
superfiles		
corrected access violation.....	43	
SYSMON		
see SystemMonitor	49	
system queues		
empty queue hang	43	
error codes	44	
incorrect return values	25	
SystemMonitor	49	
T		
terr (8989)	42	
Transaction Highwater Mark utilities	54	
U		
Unicode		
incorrect key segments	47	
USEG_ERR (707).....	25	
USERPRF_PTHTMP		
duplicate key erros	49	
Utilities		
ctcfgset.....	55	
ctclntrn.....	54	
ctdmpidx.....	55	
cthghtrn	54	
ctrbldif.....	53, 54, 56	
ctrdmp	53	
c-treeSQL		
dbdump	54	
dbload.....	54	
ctstat.....	54	
ctstop.....	55	
sa_admin.....	55	
V		
VDLK_ERR (146)	44	
vtclose	47	
W		
Windows		
Windows 2000 server crash.....	18	
X		
XCRE_ERR (669)	53	
Z		
ZREC_ERR (29)	48	