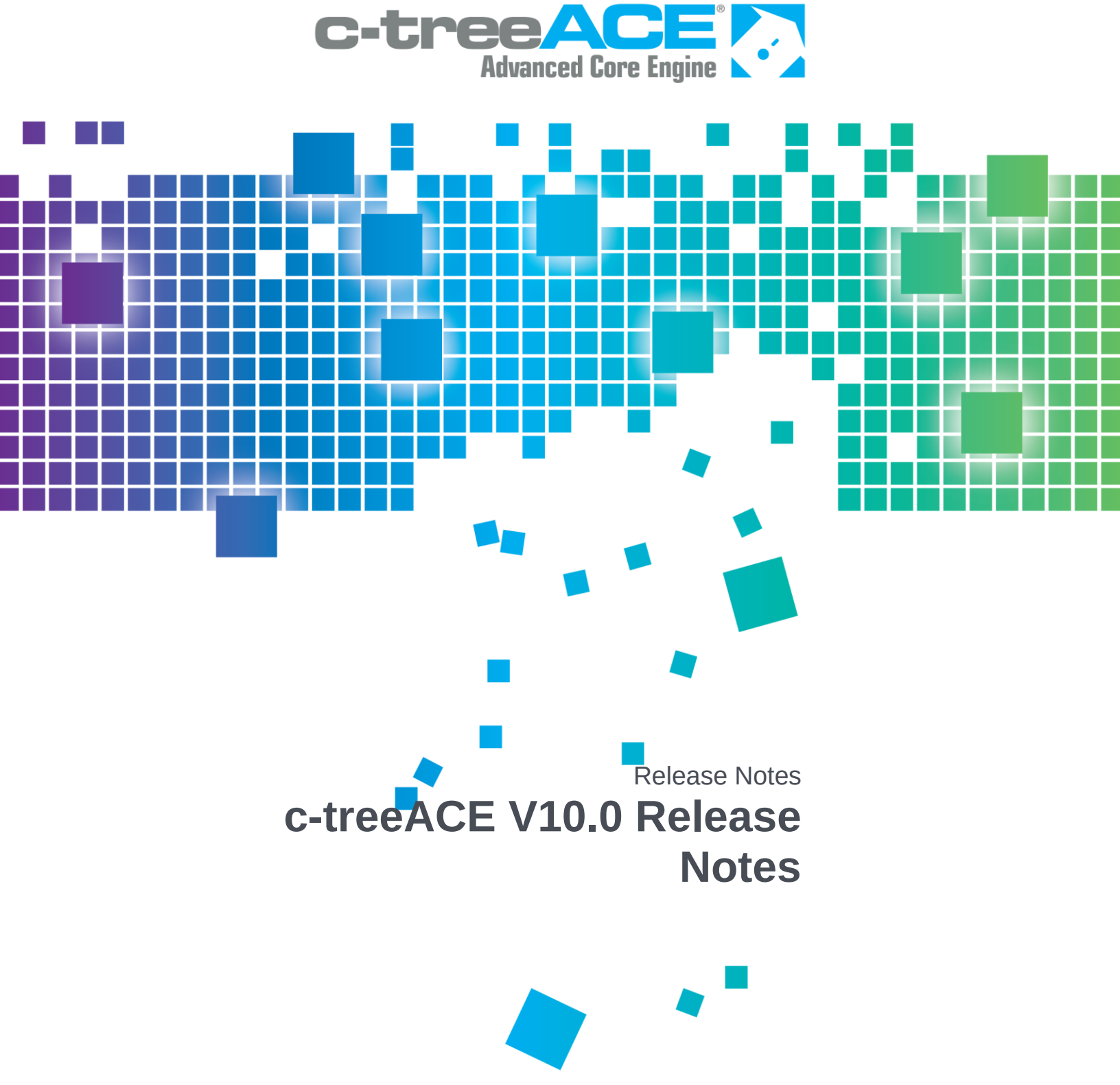

Release Notes

c-treeACE V10.0 Release Notes

Contents

1. V10.0 Release Notes

FairCom Typographical Conventions

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

The following formatted items identify special information.

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

2. Critical Production Updates

This section refers to c-treeACE updates that can directly impact uptime availability for existing production environments. These exceptional issues are specifically noted here as they have potential to affect production reliability.

2.1 Corrective Checkpoint Logic for Buffer/Cache Pages Missing from Update Lists

The c-treeACE Server had been observed to retain an excessive number of transaction logs with a message similar to the following:

```
Tue Sep  1 00:05:55 2009
 - User# 00122 The number of active log files increased to: 28
Tue Sep  1 00:05:55 2009
 - User# 00122 Cache/Buffer Pending Flush
```

This message indicates that a buffer page has not been flushed and remains on the server's internal list of updated buffers. While this can be expected in normal processing under a large transaction load, two unexpected buffer states have been considered as potential candidates for a "false" retention of the transaction log:

- The buffer does not have its update flag set.
- The buffer is not on the list of updated buffers.

c-treeACE now detects and corrects these unexpected buffer states during checkpoint processing.

In the first case, a placeholder transaction number is associated with the buffer.

To address the second case, a test is made. If a page is found off the update list for *CMTPAG_CHECK_LIMIT* consecutive checkpoints a message is logged to *CTSTATUS.FCS* and

the buffer is subsequently placed on the update list for processing during the next checkpoint. *CMTPAG_CHECK_LIMIT* is set to 6 by default.

The following is a list of potential diagnostic messages that may be seen logged to *CTSTATUS.FCS* when one of these events occur and the action taken:

```
Index buffer on commit node list without update flag...
Data cache on commit data list without update flag...
```

The c-treeACE Server found an index buffer or data cache page on the commit list, however, the buffer's update flag is not set. The c-treeACE Server enters zero transaction numbers into the commit list as this cache page has been written to disk.

```
Index buffer on commit node list is not on update list...
Data cache on commit data list is not on update list...
```

The index buffer or data cache page is on the commit node list, but not on the update list. The c-treeACE Server puts the buffer or cache page on the update list so that a subsequent checkpoint flushes that updated buffer or cache page.

2.2 Ensure Proper Mutex Control When Adding Buffers to Commit List

Internal c-treeACE QA testing observed the following messages logged to *CTSTATUS.FCS*:

```
Tue Jul 21 14:04:15 2009
 - User# 00019     WARNING: ct_cmnod adjusted  ct_cmnod:1 list_count:0 buffers:0 aux:0
Tue Jul 21 14:04:15 2009
 - User# 00019     WARNING: ct_cmnod adjusted  ct_cmnod:-1 list_count:0 buffers:0 aux:0
```

The function that adds an index buffer to the commit node list checks if the buffer is on the list before it acquires the mutex that serializes access to the list. As a result, the function might see the buffer on the commit list when it is not actually on the commit list. In this scenario, it might skip adding the buffer to the commit list, however, still increment the count of buffers that reside on the commit list.

To resolve this problem, the logic has been modified to check if the buffer is on the commit list after acquiring the list mutex. This same approach was applied to logic for handling data cache pages.

2.3 Prevent Overlapping Checkpoints

Additional synchronizations have been put into place to prevent two checkpoints from overlapping ensuring complete serialization of the checkpoint process. If the checkpoint was forced from an application level **CTCHKPNT()** call, and an in-process checkpoint is already in progress, the call returns NO_ERROR and the *sysiocod* is set to **OCHK_COD** (-885). This avoids any additional processing overhead as a checkpoint is already in progress.

A server configuration option is available to disable the check on in-progress checkpoints so they can overlap as previously allowed.

```
COMPATIBILITY CHECKPOINT_OVERLAP
```

2.4 Corrected Dynamic Dump File Extensions for Non-TRANPROC Files

A situation was observed in which c-treeACE failed during a dynamic dump and the following message was noted in the *CTSTATUS.FCS* log:

```
CTSTATUS message: getfile: 0xff fill 522 bytes at offset 0:f903f3ax
```

Due to a miscalculated size, the 'FF' fill that this message reports could write past the end of the buffer, corrupting the heap. To resolve this, the fill is now limited to the buffer size.

Note: This miscalculation only applies to non-transaction controlled files being dumped without the !PROTECT option and a non-zero !EXT_SIZE and the total dump size larger than the first segment. For transaction controlled files, it is more likely that a failed system I/O would lead to this crash condition.

2.5 Correct Initialization of c-treeACE SQL Pointers

A c-treeACE SQL crash was reported. A NULL pointer was found to cause the exception in recently added SQL IDENTITY support. While unable to reproduce this issue, it was determined that a failed pointer initialization could lead to this condition. Additional logic was added to ensure all pointers in this module are now properly initialized.

2.6 Corrected c-treeACE SQL References in an ORDER BY Clause

c-treeACE SQL could fail due to references in an ORDER BY clause being incorrectly treated as outer references. This has been corrected in the parsing phase to no longer set these as outer references.

2.7 Corrected c-treeACE SQL Conditional Expression Buffer Length Exception

An unhandled exception occurred when a SQL query read records from a variable-length table using conditions that triggered an internal conditional expression to be used. A record buffer is generally allocated based on the size of a record that was previously read from the file, and this is expected. c-treeACE logic detects any insufficient record buffer size and returns an indication that the record buffer is too small such that the caller can allocate a sufficiently-sized record buffer and re-read the record. However, the conditional expression logic that determines how many complete fields from the record the record buffer can contain did not detect the case that the final

field in the record only partially fits into the record buffer. To correct this, when the starting offset of the last field fits into the record buffer, this logic also checks that the end of the last field also fits into the record buffer avoiding the buffer overrun.

3. Notable Compatibility Changes

This section lists c-treeACE changes that affect compatibility. It is important to review these issues to ensure that your product functions correctly after upgrading to Version 10.

3.1 Latest Java Version Requirement

c-treeACE SQL V11.5 requires the Java 1.7 JDK and JRE environment for Stored Procedures, Triggers, and User Defined Functions (UDFs), JDBC, and c-treeDB Java (c-treeACE SQL V11.0 requires 1.6 or newer). Java is readily available from the *Oracle Java downloads* (*http://www.oracle.com/technetwork/java/javase/downloads/index.html*) website. c-treeACE supports Java on any platform that the Java environment is currently available, including Windows, AIX, Oracle Sun, and Linux.

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

3.2 SERVER_DIRECTORY Configuration Option Deprecated

The `SERVER_DIRECTORY` keyword has been deprecated in c-treeACE versions 9.3 and later. `LOCAL_DIRECTORY` is now the preferred keyword to allow the server to store data and files in an alternative location. The `SERVER_DIRECTORY` option did not offer robust handling of relative file paths. Full paths were the only way to ensure proper functionality. For example, with Replication and restoring dynamic dumps, a relative path caused problems in properly locating files.

To avoid potential problems with the use of this option, it has been disabled. When this option is specified in *ctsrvr.cfg*, the c-treeACE Server fails to start and displays the following message:

```
The SERVER_DIRECTORY option is no longer supported.
Use the LOCAL_DIRECTORY option instead.
```

The message is logged to *CTSTATUS.FCS*. On Unix systems it is also written to standard output and on Windows systems it is displayed in a dialog box when the c-treeACE Server is not running as a Windows service.

Use these configuration keywords to relocate transaction logs if this has been the purpose of using this keyword in the past:

```
LOG_EVEN
LOG_ODD
START_EVEN
START_ODD
```

FairCom recommends all server installations to make the switch from `SERVER_DIRECTORY` to `LOCAL_DIRECTORY` for the easiest migration path in the future. Contact your nearest FairCom office should you have any questions or concerns regarding this change.

3.3 Switch to Sbyte for TINYINT Signed Values in ADO.NET Data Provider

The c-treeACE SQL ADO.NET Data Provider was setting *TINYINT* values as *Byte* rather than *SByte* type causing a return of unsigned instead of signed values. This has been switched. ADO.NET applications using *TINYINT* values should consider this change a compatibility issue and substitute usage accordingly.

3.4 Mapping of c-treeACE SQL ADO.NET Data Provider Time Fields

The c-treeACE SQL Time type (c-tree *CT_TIME*) was originally mapped to a .NET *DateTime* type based on the lack of a Time class in the .NET CLR and that a c-treeACE SQL Time value was best represented by a *DateTime* class due to the point-in-time nature of the pair. It was noted that Microsoft mapped ODBC Time types to *TimeSpan* in .NET and SQL Server 2008 introduced a Time type that also mapped to this class. c-treeACE SQL Time is now mapped to a .NET *TimeSpan* type to better align with the Microsoft chosen mapping.

This is a compatibility change to be noted in migrating c-treeACE SQL ADO.NET Provider applications to c-treeACE SQL V9.3.

Note: The .NET *TimeSpan* class allows negative values which are not valid c-treeACE SQL time values. An attempt to update a database Time value based on a negative *TimeSpan* will result in an overflow/underflow error and proper handling of this exception should be taken.

3.5 Conditional Index Expression Changes

The Conditional Index Expression parser underwent an extensive update from V8 to V9 using the c-treeDB based parser for better integration throughout c-treeACE. Some expressions, though,

were found to give different results due to this change. As these have been identified, corrections have been put into place as described in the following sections.

Revised Integer Comparisons in Expression Parser

It was discovered that key values were missing from conditional indices composed of the following expression:

```
( field > 2148347647 )
```

Integer comparisons such as the following would also result as a comparison to -1:

```
"(unsigned) CT_INT_fld <= (unsigned)0xffffffff"
```

Date and Time types were also previously treated as integers resulting in similar failed comparisons.

Integer types less than four bytes were stored in a four-byte signed variable resulting in unsigned values cast to signed values for comparison purposes. The integer storage type has been changed from *LONG* (4 bytes) to *LONG8* (8 bytes). Expression evaluation should now match previous behavior.

Corrected CVAL_ERR on Partial Record Reads

A conditional expression failed on a partial record read with error **CVAL_ERR** (598, could not evaluate conditional expression) when all fields in the conditional expression were read with c-treeACE V9, although such an expression had previously succeeded. Version 9 introduced a new underlying architecture for the conditional expression logic. The internal **fixDodaOffsets()** function call failed if a partial record buffer was encountered. This function is now allowed to succeed on a partial read, and fails if the expression parser encounters a field that was not read. This was the behavior prior to V9.

NULL Handling in Filter Expressions

Prior to V9, records values with the NULL attribute set are returned even when a filter was in effect. Beginning with V9, these records do not pass this filter condition. V9 now checks if a field is NULL and treats a NULL value as distinct from a value of 0.

3.6 TCtDataSet Fields Property Type Has Changed in c-treeACE VCL

The c-treeACE VCL *Fields* property of the *TCtDataSet* class had overwritten the generic *Fields* property of the *TDataSet* class (the parent of *TCtDataSet*). c-treeACE VCL used a type of *TCtField* while the parent field type was of *TField*. As the *TCtField* type is meant for internal use, it has been renamed *CtField*.

As a result of this change, c-treeACE VCL now has the default generic *Fields* property from *TDataSet* available in *TCtDataSet* and there are two ways to retrieve field information:

- *TCtTable.Fields[<field index>]* works exactly as *TCtTable.FieldByName(<field name>)*

- With *TCtField*, specific information remains available as *TCtTable*.*CtFields[<field index>]*

3.7 V9.1 Client/Server Compatibility

V9.2 introduced a modified version of the FAIRCOM.FCS user definitions. As a result, prior client server compatibility can be affected.

If a client prior to c-treeACE V9.2 tries to open the file *FAIRCOM.FCS!USER.dat* with a c-treeACE Server with an updated *FC_USER* structure, the open will fail with error **UVRC_ERR** (859, The client's structure definition for the file *FAIRCOM.FCS!USER.dat* is out of date. Update your client library.)

If a V9.2 or later client with with an updated *FC_USER* structure attempts to open the file *FAIRCOM.FCS!USER.dat* using a c-treeACE Server without the updated structure, the open will fail with error **UVRS_ERR** (860, The server's structure definition for the file *FAIRCOM.FCS!USER.dat* is out of date. Update your c-treeACE Server.)

3.8 Rollback and Commit Support in c-treeACE ADO.NET Data Provider

When the ADO provider was first developed, logic was added, with an exception thrown, that prevented a user from calling **Rollback()** and/or **Commit()** while the *DataReader* object was still active (i.e. has not been closed). However, as **Rollback()** and **Commit()** close the *DataReader* rendering it unusable, it was determined this could be problematic as a **Rollback()** may be called by code that does not intend to still use the *DataReader* after the **Rollback()** (and, in any case, a different exception is thrown). This additional check, and the related exception, has been removed.

4. c-treeACE Fixes

4.1 Corrected Inappropriate DAR RBUF_ERR on File Opens

It was found that the Direct Access Resource (*DAR*) initialization caused a file open to fail with a **RBUF_ERR** error (404, output buffer too small) after adding a resource larger than a *DAR* resource size. When c-treeACE opens a file, it reads the *DAR*s from the file and processes them. If a standard resource follows the last *DAR* in the resource chain and the size of that resource exceeds the size of the *DAR* structure, the call to read the resource returns the **RBUF_ERR** error, resulting in a failed file open. Now, when reading the *DAR*s, if the call to read the next resource returns error 404, the resource type is checked, and if it is not a *DAR*, reading is stopped and NO_ERROR is returned.

4.2 Faster Server Shutdown with Large Memory Allocations

The c-treeACE database engine normally frees associated memory (data and index caches) when shutting down if all clients have disconnected. However, it was discovered that freeing this memory can take a long time when using very large data and index caches due to the overhead of returning the allocated memory blocks to the process heap. When the c-treeACE process is about to exit, returning the memory to the process heap is not necessary. This housekeeping activity is now skipped when shutting down to avoid observed delays.

4.3 Enhanced Windows Service Shutdown Operations

Windows Vista and later allow a new service *PRESHUTDOWN* notification. With normal *SHUTDOWN* notification, the service will be killed if it takes more than approximately 90 seconds to stop. When a service signals that it handles a *PRESHUTDOWN* notification, it will be given a greater amount of time to complete its shutdown. This can avoid a lengthy auto-recovery should the server have a large number of cache pages to flush at shutdown, such that it cannot complete under the *SHUTDOWN* limit. c-treeACE now enables handling of the *PRESHUTDOWN* notification on the selected Windows versions.

4.4 Corrected Ability to Kill Dynamic Dump Threads

It was discovered that it was not always possible to terminate a dynamic dump thread waiting for a scheduled dump time. The dump thread did not appear to terminate and/or the **ctadmn** administrator utility request to kill a thread reported error **SUSR_ERR** (97). When the dynamic dump thread was sleeping until the scheduled dump time, and was awoken by a request to terminate the dump thread, a subsequent freeing of the thread's state variables cleared an internal indicator. This indicator is used to cause the dump thread to properly complete the termination process. A copy of the state variable is now made before it is cleared resulting in the

subsequent test to succeed, and the dump thread is properly terminated. The **SUSR_ERR** was the result of a timing issue in detecting the thread's state variables, which may or may not have been freed at the time of the check. This too has been modified to avoid the unnecessary error.

4.5 Deferred Transaction File Number Assignment

While investigating a case of "pending transaction file ID overflow" messages, it was found that transaction file numbers could be deferred until the file was actually updated. Previously, when c-treeACE physically opened a transaction-controlled data or index file, it assigned a transaction file number (*tfil*) to the file and wrote an OPNTRAN entry to the transaction log buffer. As a result, even if a file was simply opened, read, and then closed, the file consumed a *tfil* value and logged OPNTRAN/CLSTRAN entries. Now, c-treeACE defers the *tfil* assignment and OPNTRAN logging until the first update is made to the file. The logging of the LTFLMAP entry for a replication unique index is also deferred in this same manner.

4.6 c-treeACE now Skips Duplicate SetNodeName Transaction Log Entries for Faster Performance

While analyzing transaction logs for automatic recovery performance, multiple duplicate entries for setting a client node name were found. To avoid these numerous entries, the client-side logic for the **SetNodeName()** function now checks if the specified node name is the same as the current node name for the connection. If so, **SETNODE()** returns without making a call to c-treeACE and avoids the extraneous LLOGNODE entry in the transaction logs.

4.7 Permit Shared Reopen After a Transaction Controlled Header is Updated

Occasionally, it is quite useful to update header information in a file after it is created and before any users begin using the file. Consider the case of setting an initial serial number.

If a file is opened exclusively, and a **PUTHDR()** call updates a file attribute (such as the next serial number) under transaction control, and the file is closed and then reopened in shared mode before the transaction is committed or aborted, the following scenarios could apply:

1. Prior to this change, the reopen would fail with **FNOP_ERR** (12) / **FCNF_COD** (-8);
2. With this change the reopen succeeds, however, other users still see the file opened exclusively until the transaction is committed or aborted.

With the introduction of this open modification, other shared open attempts fail with **FNOP_ERR** (12) / **FCNF_COD** (-8) as if the file was still in exclusive mode. Once the transaction is completed (commit or abort), then other shared opens behave as expected.

4.8 Allow Automatic Recovery of Uncommitted TRANDEP Files

An unexpected **FNOP_ERR** (12) from AUTO SKIP of missing files was reported in a server automatic recovery process. Newly created transaction dependent (*TRANDEP*) files were in a checkpoint before the final disposition of the files was determined. In this case, after the files appear in the checkpoint their create was aborted and the system terminated abnormally. Recovery found the files in the checkpoint but did not see any *TRANDEP* activity in the log (even though log entries are present they had not yet been read). This led to a report of (auto skip) missing files. In this case, the problem was that the checkpoint entries for the files did not contain any indication that the files have just been created and not yet committed. The contents of the checkpoint have been augmented to include the additional information related to *TRANDEP* file creates which will allow recovery to proceed as usual, even in the case on uncommitted *TRANDEP* files.

4.9 Correct Transaction Log Encryption Incompatibility

When the `LOG_ENCRYPT` and `ADVANCED_ENCRYPTION` configuration options were specified in *ctsrvr.cfg,* the c-tree Server failed to start with a log incompatibility error **LFRM_ERR** (666, lfrm = 5), even if the server was started in a directory with no existing transaction log files.The transaction log file compatibility check failed to account for the log encryption option in effect when using advanced encryption, and assumed the transaction log files use a feature not supported. This check has been adjusted to account for the advanced encryption option.

Additional comprehensive checks at server startup have been added to ensure that the c-tree Server log encryption settings are able to handle the type of encryption for existing transaction logs. When an incompatibility is found, server startup now fails with an error message in *CTSTATUS.FCS* indicating which options must be changed to access the transaction log files.

The table below shows the expected results for the possible combinations of the `LOG_ENCRYPT` (LE) and `ADVANCED_ENCRYPTION` (AE) configuration options for each possible transaction log encryption type used by an existing log file (LogEncTyp).

LE	AE	LogEncTyp	Expected result
N	N	none	Successful startup
N	Y	none	Successful startup
Y	N	none	Successful startup, new logs are scrambled with CAMO*
Y	Y	none	Successful startup, new logs are encrypted with advanced encryption
N	N	camo*	Error: Enable log encryption to proceed
N	Y	camo*	Error: Enable log encryption and disable advanced encryption to proceed
Y	N	camo*	Successful startup
Y	Y	camo*	Error: Disable advanced encryption to proceed

LE	AE	LogEncTyp	Expected result
N	N	advanced	Error: Enable log encryption and advanced encryption to proceed
N	Y	advanced	Error: Enable log encryption to proceed
Y	N	advanced	Error: Enable advanced encryption to proceed
Y	Y	advanced	Successful startup

* CAMO or "Camouflage" is an older, legacy method of hiding data, which is not a standards-conforming encryption scheme, such as AES. It is not intended as a replacement for Advanced Encryption or other security systems.

4.10 PRIME_CACHE and PRIME_INDEX Configuration Options Parsing Corrected

It was found that specifying the c-treeACE configuration options `PRIME_CACHE` or `PRIME_INDEX` with values greater than 2 GB, the cache was not primed with the specified number of bytes of data records or index nodes. Switching the cast of the variable used for the numerical value to an unsigned value corrects this deficiency.

In addition, the scaling factors (for example 2GB) failed to operate on the `PRIME_INDEX` keyword. Extending the scaling factor logic now adds this capability.

4.11 Resolved Filter Callback Function 160 Errors

The filter callback feature provides a mechanism for a developer to apply custom filtering logic for c-tree data record reads and writes. Enhanced support was enabled that allowed c-tree calls from within this callback logic. It was possible to encounter an **ITIM_ERR** (160) error when using these callback filters causing them to fail.

An internal key buffer used by the record read was overwritten by a call to read a record in the filter callback function's code. (The logic that reads the record in the filter callback function is written by the application developer.) After the filter callback function returned, c-tree compares the key value formed from the record contents with the saved key buffer contents. However, the key buffer contents had been changed, resulting in the 160 error.

To avoid this, the contents of the key buffer are now saved before calling the filter callback function, and the contents of the key buffer are restored after the filter callback function returns.

4.12 Corrected Hangs with Data Filter Callbacks

When a client causes the c-treeACE process to load a data record filter callback DLL, if the client loses its connection to the server without first disconnecting, the server thread for that connection might hang when it closes the files for that connection if the data record filter callback DLL is used

in an ISAM context and the data record filter callback DLL closes a file in its unload function. When this situation occurs, the symptoms are:

1. clients hang when attempting to connect to the server, and
2. the message "tflstt loop alert" appears in *CTSTATUS.FCS* after about 90 minutes.

When c-treeACE closes the data and index files that are in use by a connection, it closes any ISAM contexts that have been established for the files. If a data filter is in effect for an ISAM context, the filter is freed. If the data filter uses a data filter callback DLL, the freeing of the data filter causes the filter DLL's unload function to be called. If the filter DLL's unload function closes the file that is in the process of being closed, the thread can deadlock itself by waiting for a temporary file state to clear (which will never happen because it is the thread itself that set that state).

Now, when an ISAM context is being closed and the data filter resources for that context are being freed , we check if this is happening while closing a file. If so, we defer the freeing of the data filter resources until the file close completes.

4.13 Resolved c-treeACE Unhandled Exception from Failed Filter Callback Library Loading

It was possible, when reading a record after a user-defined data record filter callback DLL initialization function returns an error, the c-tree Server process terminated with an unhandled exception. When the **SetDataFilter()** function loads a data record filter callback DLL and the load library callback function exported by the DLL returned an error, **SetDataFilter()** unloads the DLL, however, did not disable the use of the data filter. As a result, when a client read a record, and attempt to call the data record filter callback function was made with a NULL pointer address. **SetDataFilter()** was modified to disable the use of the data filter when the setting of the data filter fails because the load library callback function returns an error.

4.14 Persistent Transaction Lock State now Ignored for Unlock Requests Outside Transactions

It was reported the *ctLOKDYN* mode behaved in an unexpected manner regarding the persistent lock behavior for transaction-controlled files previously implemented. Essentially, **ctLOKDYN(***tranPersist***)** also changed the behavior of locks outside of transactions and it was preferred to have locks acquired and released entirely outside of a transaction to be immediately released. Currently, they are held until a **TRANBEG()** + **TRANEND(***ctFREE***)** is called if the *tranPersist* mode is set. This behavior change was not mentioned in the original implementation for the persistent lock feature.

The function that unlocks a data record was modified such that it now rejects an unlock request for a record in a transaction-controlled file when the persistent transaction lock state is enabled only if a transaction is active at the time the unlock request is made. This means that an unlock request that is made outside a transaction will ignore the persistent transaction lock state.

4.15 Correct Free of All Locks with Persistent Transaction Locks Mode

If a client calls **ctLOKDYN(***ctLOKDYNtranPersist***)** to enable persistent transaction locks and a variable-length record is updated and grows in place, when the file is closed the c-tree Server logged the message "Unclaimed locks found" to *CTSTATUS.FCS*.

The **RewriteVRecord()** logic locks the starting offset of the space into which the record is growing, updates the record header to indicate the new record length, and deletes the key from the space management index that points to this space that has been reused. After these operations are completed, reused space is normally unlocked. However, when using the persistent transaction lock feature, this unlock takes no action as a connection-specific state variable was set to indicate that transaction locks are to be persisted. As a result, that space remains locked until the file is closed, at which time the c-tree Server finds the lock in the system lock table for that data file and logs the "Unclaimed locks found" message to *CTSTATUS.FCS*.

This lock on the reused space is an internal lock that should always be freed when the update to the record header and space management index have been completed. **RewriteVRecord()** was modified to correctly handle the persistent locks mode in the state variable by clearing the state bit upon entry into the function and re-enabling the state on exit.

4.16 Removed Savepoint COUNT Limitation

A **RVHD_ERR** (123) was observed when reading a record from a variable-length data file after the record was updated in a transaction consisting of many operations separated by savepoints. After the update, in the data file only the record body (no record header) existed at the record offset. The transaction had been comprised of more than 32768 savepoints, however, the function that searched preimage space for an entry matching the target record offset cast the savepoint value to a *COUNT*. As a result, a preimage space entry containing the record image for the current savepoint was not found and a new entry containing only the record body was added to preimage space. Removing the data type cast in the assignment of the savepoint number to a local variable resolves this issue.

Note: There still remains a *COUNT* size limitation on the return value of the **SetSavepoint()** API call. Due to this restriction, it is advised to not specify more than 32768 savepoints within a transaction. Using the special rolling savepoint through the **ReplaceSavepoint()** API call avoids this restriction.

4.17 Improved Use of Thread Safe Calls

It was found that certain system calls may not be thread safe. Several system calls are made during a dynamic dump when using wildcards to specify files. Specifically, the following calls were reviewed and replaced with thread safe versions when appropriate:

- **getpwnam()**
- **readdir()**
- **getgrgid()**

- **getpwuid()**

4.18 Corrected Data Length of CTUSER() Custom Output Data

An output data length set by the **CTUSER()** function did not limit the amount of data returned to the client. Recently applied **CTUSER()** changes resulted in the output length set by **CTUSER()** to have no effect on the amount of data returned to the client. This is because the communication buffer was set to NULL before calling **CTUSER()**.

To resolve this, the internal name of the **CTUSER()** function was changed in *ctsctu_a.c* to **iCTUSER()** and a parameter was added that is a pointer to the output length. **iCTUSER()** returns the output length in this parameter and logic was added to return a proper length of data.

4.19 Correct Resource Updates at EOF

A table open failed with a **RRED_ERR** (407) following a c-treeDB ALTER TABLE operation. The **UpdateResource()** function allows a resource to grow in place if it is located at the end of the file. An internal calculation to determine the end of file failed to properly account for a 2GB boundary due to a signed value. This caused anything following the resource to be improperly overwritten. Changing the value to an unsigned value before the comparison corrects this error.

4.20 Corrected Shared Memory Client Hang on Windows

A client request to c-treeACE was found to hang when using the shared memory protocol. The server thread for that connection had already exited. The server can encounter an error situation before it reads a request from the client, and in that case it did not send a response to the client. The server thread closed its shared memory connection while the client still has open handles to the interprocess semaphores and shared memory region. As a result, it did not detect the server thread closing the connection. (By contrast, when using the TCP/IP communication protocol in this situation, the server thread closes the socket, which the client can detect.) An example case that demonstrated this problem, a dynamic dump (with the !DELAY option specified in the dump script) caused a transaction to be canceled.

A connection state bit is now set to indicate to the shared memory logic that it should set a flag in the shared memory region that notifies the client that the server thread has closed the connection. The client is now expected to receive error 128 in these situations (instead of waiting forever, hanging, in the read request).

4.21 Corrected Automatic Recovery of Files with TRANDEP Renames / Deletes

An issue was discovered involving automatic recovery of transaction dependent file renames and/or file deletes which were pending commit when the server terminated abnormally. This issue has now been corrected. A pre-scan of the log looks for certain uncommitted entries and compiles a list which is used when automatic recovery processes the log.

5. Partitioned File Fixes

5.1 Reinitialize Arrays to Prevent Unhandled Exceptions When Opening Partitions

When a client reads a record from a partitioned file, and if reading the record requires the c-treeACE to open a partition, an unhandled exception could occur. Opening a partition might require resizing an internal global array and this resizing changes the base address. However, some local variables no longer point to the correct element for the host index. Reinitializing these local variables resolves the incorrect dereferencing.

5.2 ALTER TABLE now Disables Transaction Support for Partition Files

When a SQL ALTER TABLE operation is used to create a partition index on a large table, c-treeACE SQL could terminate with an out of memory error, such as a terr(7496) or **CMEM_ERR** (602). The alter table operation normally creates files with transaction control disabled to avoid the performance overhead of transaction logging. However, when a new partition is created, c-treeACE closes and reopens the partition, which caused transaction logging to be enabled for that partition.

When transaction control is active and records are added to the file, the records are written to pre-image space until the transaction is committed at the end of the alter table operation. For a large file, this will result in significant memory use and possibly exhaust memory depending on the size of the data file. Logic has been modified to specifically avoid the transaction control for alter table operations on partitioned files.

5.3 Delete all Partitions When Deleting the Partition Host File

Previously, an SQL drop of a partitioned table deleted the host file but did not delete the associated partitions. This modification introduces the full deletion of all partitions associated with the table. The individual partitions are now deleted before the host file is removed.

5.4 Improved Internal Error Checking on Partitioned File Opens

An abnormal server termination suggested an internal call unexpectedly failed , however, the error return was ignored in a subsequent call, resulting in a segmentation fault. While we have

been unable to construct a scenario that would lead to this situation we found we need to enforce the check on this return value which should prevent dereferencing an invalid pointer value.

5.5 Drop of non-Partition Index Files for Partitioned Tables in c-treeACE SQL

Previously, an attempt to drop a partitioned index that was not the partition key had no effect, as this operation was not supported at the SQL level. The logic has been adjusted to now properly drop non-partitioned indexes for partitioned tables.

5.6 Corrected Record Return for Range Retrieval Expressions on Partitioned Files

A SQL query on partitioned files failed to return some records that should match the query criteria. A range retrieval with a filter expression on a variable-length record failed with error **CVAL_ERR** (598) as the buffer was not large enough to hold the portion of the record that is required to evaluate the filter expression. This is a normal situation, and the logic that called the range retrieval function detects the error and reallocates the buffer such that it is large enough to hold the record. The expectation is that calling the range retrieval function again will return the same record. However, a recent optimization to the partitioned file key buffer did not consider this particular situation, and the next key lookup for the partition whose key was returned on the previous range retrieval (and whose record failed the range check with **CVAL_ERR**) read the next key from the index instead of using the key from the partition file key buffer. To avoid this missed buffer read, when a **CVAL_ERR** occurs on a partitioned file that is using the partition key buffer, the partition key buffer state is set for that partition indicating it is necessary to read the key from the partition key buffer.

5.7 Correct File Modes Applied to Partioned Files During ALTER TABLE

While attempting to drop a column and/or index, it was noted that a SQL ALTER TABLE operation on a partitioned file removed the *TRNLOG* and *TRANDEP* attributes from the partitions although the partition host retained these attributes.

An ALTER TABLE operation on a partitioned file creates a new set of partitions. The host and partitions are initially created without *TRNLOG* and *TRANDEP* for performance considerations. At the completion of the ALTER TABLE, calls to **PUTFIL()** and **PUTHDR()** are made on the partition host files to enable the *TRNLOG* and *TRANDEP* modes on the files. However, these calls only affected the partition hosts and the newly-created partitions were not updated.

To ensure all partitions are assigned proper file modes, when **PUTFIL()** is now called on a partition host file and **PUTHDR()** is called with a mode of *ctXFL_ONhdr* on a partition host file, these functions now also apply the corresponding filemode changes to all existing partitions for that host.

Also, when an ALTER TABLE operation is performed on a partition host file that has existing partition member files, the partition members were recreated using temporary names instead of the original name. SQL is able to access the files, but it is difficult to identify in a directory listing which partition member files belong to which partition host file.

To make this clearer, **RENIFIL()** was modified such that for a partitioned host file all existing partition members are renamed and the partition host member resource is updated to reflect the new partition member file names.

5.8 Complete Record Retrieval with Non-Contiguous Ranges on Partitioned Files

A range operation on a partitioned file involving criteria that match non-contiguous keys omitted some records that match the range criteria. In the case involved, the range consists of two non-contiguous ranges. First, a key is read and this key is checked against the second range criterion, which it fails. The range is then promoted to the first range and in this case should find the same key and the key passes the first range criterion. However, when the partition key buffer is enabled (default), the key buffer should be ignored in this situation and the key search performed. To ensure the key search is performed, c-treeACE now resets the partition key buffer state so that the key search is performed when switching between range limits in a non-contiguous range.

5.9 Provide Connected User List for Partition Administration Exceptions

A partition purge was failing with **PUSD_ERR** (718, partition member in use); however, connections were not expected to have the partition member open. Diagnostic logging was added to better diagnose which partition and which connection were involved.

Now, when a partition member purge fails with **PUSD_ERR**, the filename and list of open instances of that file are logged. Below is an example. For each connection, the task ID, user name, node name, and user file number are listed.

```
Mon Dec 12 12:40:33 2011
 - User# 00012  PT_ADMIN: purge failed, partition .\ctreesql.dbs\admin_pt.20111129.015307.dat is
open (2 open instances):
Mon Dec 12 12:40:39 2011
 - User# 00012  PT_ADMIN: Connection 16: ADMIN(SQL:CTREESQL) 64
Mon Dec 12 12:40:44 2011
 - User# 00012  PT_ADMIN: Connection 17: ADMIN(SQL:CTREESQL) 45
```

In addition to these log messages, if the configuration option `DIAGNOSTICS PTADMIN` is specified in *ctsrvr.cfg*, then c-treeACE creates a process stack trace when this error occurs. This option can be enabled or disabled using the **ctadmn** Administrator utility.

5.10 Correct Index Name Reported for Duplicate Key Errors on Partitioned File Global Unique Indexes

A duplicate key exception was occurring during a record insert, however, the index reported was not the actual index with the duplicate key error. When unexpected duplicate key errors were returned on a global unique index (GUIx) the data file number (*datno*) was returned rather than the actual index file number (*keyno*). Correct file information is now back propagated to report the proper index name.

5.11 Correct Rebuild of Covering Indexes for Partitioned Files

A situation was found in which the partition file index contained no keys after deleting the indexes and rebuilding. This occurred for a "covering index" that included the partition key, and was not ordered as the partition key. In this specific case, the rebuild terminated before rebuilding the indexes, and returned no error. The logic assumed the indexes were open, but at this point of the rebuild they were all closed, causing **FACS_ERR** (26, file number not in use) which was not propagated. Rebuild logic was altered to set up ISAM-level relationships between the covering index and the partition key to correct this.

5.12 Correct Transaction Logging of Partitioned File Serial Segment Entries

After automatic recovery, record inserts unexpectedly failed with duplicate key errors on the c-treeACE SQL *ROWID* index. and these eventually succeeded after several retries. It was determined a transaction log entry was missing under specific circumstances of an update to a partitioned file serial segment value, used by the internal *ROWID* field. These files had recently undergone automatic recovery to to a prior server crash resulting in the header value of the file not in sync with the highwater mark of the file. This has been corrected by ensuring the log entry is correctly written.

Note: While this involved automatic recovery, data and index values maintained integrity. Only the serial segment file header value was affected. The symptom would be unexpected duplicate key entries, and eventually succeeding after several attempts as this value is incremented.

5.13 Corrected Unhandled Exception in Partitioned File Search Routine

The partitioned file key search routine did not properly handle a case in which the logic unexpectedly returned a NULL, resulting in a server crash. The logic was modified to correctly handle this situation and log information to *CTSTATUS.FCS.* While modifications were made to

correctly handle the NULL condition, this logging will help identify the underlying problem should it re-occur. Below is a sample of the information now logged in *CTSTATUS.FCS*:

```
Wed Feb 29 15:30:23 2012
 - User# 00011  Failed reference to partition index file. Error# 26
                Raw Partition# 13  Relative key# 2
Wed Feb 29 15:30:23 2012
 - User# 00011  Partition data file# 4   Partition index file# 7
                User file# high watermark 12
Wed Feb 29 15:30:23 2012
 - User# 00011       Host data file# 0   Host index file# 3
Wed Feb 29 15:30:23 2012
 - User# 00011  Partitioning index relative key#: 0
                Partitioning index file#: 1
Wed Feb 29 15:30:23 2012
 - User# 00011  vcusti
Wed Feb 29 15:30:23 2012
 - User# 00011  vcusti.2
Wed Feb 29 15:30:23 2012
 - User# 00011  vcusti.013
```

In addition, additional modifications were made to prevent unexpected internal **FACS_ERR** (26) errors. It is possible for a host file to stay physically opened while a partition file is deferred closed. A subsequent open of the host causes it to be logically opened, however, an internal state was not maintained as the file remained physically opened. The search routine detected the file was closed and returned a NULL file control block (FCB) and setting the error to **FACS_ERR**. This NULL FCB can then lead to an unhandled exception if dereferenced. This is now prevented by adding a check for deferred close states.

5.14 Diagnostic Logging of PCRP_ERR (724) and KLNK_ERR (25) Errors

PCRP_ERR

PCRP_ERR (724) is generated when the current context of a partitioned file is not correct. For example, rewriting a record without a prior read record, or performing a next record operation without a prior first or equal record operation. To help explain unexpected **PCRP_ERR** errors, a configuration keyword, `DIAGNOSTICS PCRP_ERR`, can be specified in *ctsrvr.cfg*. enabling diagnostic logging when a **PCRP_ERR** (724, bad current ISAM position for host) is returned. A message is logged to *CTSTATUS.FCS* and creates a process stack trace (minidump) of the process when this configuration is active.

An example message is shown below:

```
PCRP_ERR: ctpartno(), file <filename>
PCRP_ERR: ctunfoldpartno(), file <filename> loc <location>
PCRP_ERR: subno=<subno> dnum->ptcur=<ptcur> dnum->ptmbr=<ptmbr>
```

KLNK_ERR

A **KLNK_ERR** (25) results when an internal index node link is determined incorrect, or missing. As this is a very rare occurrence, it can be extremely difficult to isolate the conditions leading to this error. When a **KLNK_ERR** error now occurs, a stack dump of the c-tree Server process is

created (at most once every 5 minutes) to provide a complete picture of the state of the system at that time.

5.15 Improved Partition Purging

A partition file purge could unexpectedly fail with error **PUSD_ERR** (718). While attempting to reproduce the issue reported from the field, it was found an internal server hang could occur. A partition purge attempts an internal file block call to allow an open partition member to be closed while holding a mutex for the partition host. If a thread is waiting for the same host file (for other operations such as closing a partition member), it was unable to complete without acquiring the same mutex.

The timing of this situation could result in either a file block timeout, or deadlocked threads, resulting in a server hang. Evidence of this occurrence was suggested with following reported server logging:

```
- User# 00020      ctFILBLK Note: no progress clearing threads from core. Abort block attempt.: 842
```

To avoid this condition, the mutex is now released before calling the file block and repeat the mutex acquisition after the file block completes.

5.16 Prevent c-treeACE SQL c-treeDB Internal Errors From Reused Partitioned Members

A case was presented that exhibited a c-treeDB internal error via the SQL engine. It was determined this occurred as a result of a partitioned member being reused after purging.

A c-tree file header update routine used to enable the partitioned file *IDXENT* search optimization (*ctGUIxIDXENThdr*) properly detected purged partitions, however, did not take into account partitions that had been purged and made available for reuse. Typically, reuse is useful when it is known that data will be used to fill the partition member again, such as occurs during application testing. The header update now performs additional checks to always determine the existence of any reused purged partition members.

5.17 Retain Purged Partitioned Member Information of Reused Partitions

After calling **RenameIFile()** on a partition host file that has partition members that have been purged and made reusable (but not yet recreated), the partition resource in the partition host file does not contain entries for these partition members. This can potentially cause problems if a global unique index exists and it contains key values from the purged partition members, as removing the partition member entry from the partition resource loses the partition instance information for that partition. The partition instance value is part of the key value in the global unique index to determine which partition a key belongs to. To prevent this loss of information, the entry for the partition is now kept regardless, when a partition instance value is non-zero.

5.18 Improved c-treeACE SQL Index Choice for ANL or Sort Based on Cardinality

While investigating partitioned file query performance, certain queries were identified to run in sub-optimal time, even when not partitioned. It was found that the optimizer would prefer an index for sorting rather then to optimize the query, thus resulting in slower performance. Forcing the join order with the "ctree ordered" clause was found to revert this condition.

Priority was almost always given to sorting when choosing an index and hence fix the join order such that tables with the sort columns are placed on the left (LHS and RHS switch case candidates are not considered during join method optimization). In some cases, it was found that choosing the correct join order based on ANL condition could lead to very low cardinality from the join node and hence better performance even if the sort node is retained in the plan.

For the switch case candidate, check if the sort hint was set for this join node and the sort columns are also from this node. If so, check if the cardinality of this join candidate is less than 1000, and also if the cardinality is 20 time less than the cardinality of the currently selected join candidate. If this is the case, allow the current candidate to be considered as one of the candidates, otherwise ignore it.

Note: Currently this will only allow switching when the sort columns are NOT from multiple tables.

5.19 Correct Unhandled Exception When Accessing Partitioned File Before Partition Members Have Been Created

An unhandled exception could occur if partitions members had not yet been created and the host was accessed. As no partition member files yet exist, an internal array had not yet been allocated and there were two cases where this would cause exception. The recent addition of the GUIx key count optimizations made falling into these cases more likely with files access via SQL. These situations are now avoided with appropriate handling of the initial values of internal partition management variables.

6. c-treeACE SQL Fixes

6.1 Corrected c-treeACE SQL Memory Leaks

While profiling a c-treeACE SQL Server environment, it was discovered the process increased memory usage over time under specific conditions. Several areas were profiled in depth for memory utilization and potential for memory consumption was addressed:

- An **ExecuteDirect()** ODBC function call was found to cause memory leakage when a SELECT statement timed out, and the statement was not properly closed. To prevent memory from being returned, failed statements are now closed.
- When a c-treeACE SQL client disconnected, a small (99 byte) leak was found in early versions of c-treeACE SQL 9.3.
- Multiple SQL parsing and optimization routines failed to release internal structures resulting in memory leakage over time. These have been identified and corrected.

6.2 Additional Diagnostic Logging of SQL Errors

Some SQL error conditions return information that can appear somewhat generic, and as a result, makes diagnosing particular issues difficult. To enhance diagnostic capabilities, when a SQL error now occurs, a new error logging option is available.

```
SQL_DEBUG ERROR_INFO
```

When this keyword is enabled in the server configuration, and a SQL specific error occurs, an event is logged in the *sql_server.log* file in the server's working directory. Information included in this event include any error that may occur running a query, the query statement, the execution plan, and the SQL RPC function number that returns the error. This will provide a valuable source of information in the case of internal handling errors of SQL, such as the MM susbsystem for the storing and sorting of temporary tables.

6.3 Correct Results from Nested TOP Queries and ORDER BY Clause

A query with a subquery using TOP and ORDER BY on an indexed field returned an incorrect result. It was found that a sort optimization did not properly track that an index could have been used to perform the sort. An optimization routine has been adjusted to take into account all available indexes for proper sort ordering.

6.4 Consistent Results Using '+' as String Concatenation Operator

A query using the "+" operator to concatenate strings could interfere with padding of CHAR types. The "+" operator is supported to concatenate any CHAR types. However, when a VARCHAR is concatenated with a CHAR using "+" and the second operand is CHAR the resulting output type is CHAR and contains extra spaces due to padding issues. (When the second operand is VARCHAR, the output type is VARCHAR and there are no spacing issues.) The "||" operator is also documented as a string concatenation operator and using this construct returns proper results. When using the "||" operator or CONCAT, the output type is always VARCHAR, and the issue doesn't arise. The "+" operator has been modified such that it behaves in the same manner as "||" or CONCAT.

6.5 Proper View Column Names

Returned column names for a view were found to not match the actual column names. When the column names were not explicitly specified, a logical error was encountered. When a '*' is used in the SELECT clause of a view definition, the order of the expressions in the global list was assumed to be in the same order as the columns referred to in the SELECT clause. This assumption was incorrect, and appropriate column names are now returned for the view.

6.6 c-treeACE SQL Java Runtime Engine Handling of Stored Procedure JAR

Calling a Java stored procedure, trigger, or User Defined Function when the *ctreeSQLSP.jar* file was not available terminated the server. While the server could correctly start with the Java runtime engine loaded, the stored procedure class file could be referenced without actually being present. A proper check is now made to ensure this class is available, and if not, a **SQL_ERR_JSPTENOTSUP** error is returned.

6.7 Proper c-treeACE SQL User Defined Function Exception Handling

When throwing an exception from a User Defined Function (UDF) in c-treeACE SQL the server could fail due to a NULL pointer reference, as the UDF had thrown the exception before the return value was allocated. Complete exception handling has been put in place to prevent this situation.

6.8 Improved Checks on Security Calls for Index Members to Avoid WRITE_ERR

It was found that the commit of a SQL create index statement could fail with a **WRITE_ERR** error (37). A recent revision involving a call to the **SECURITY()** function caused an additional transaction entry to be added for an index member. When the transaction committed, it used the file control block of the index member rather than the index host, which resulted in the wrong file descriptor being used for the I/O.

The **SECURITY()** logic now checks if a call was made to set the file owner, group, or password for an index member, and If so, returns an **FMOD_ERR** error (48, operation incompatible with type of file). For c-treeDB alter table operations, including SQL index creations, the **FMOD_ERR** error is ignored allowing the **SECURITY()** function to effectively fail silently and the index creation operation completes without further error.

6.9 Correct c-treeACE SQL PHP Handling for Queries Returning No Result Sets

A PHP invoked query calling a stored procedure that did not return a recordset failed with error **-20039** (Invalid cursor state, open for non-select statement). The c-treeACE SQL PHP interface did not take into account these non-select statements as stored procedure were assumed to always return a recordset. However, this is not always appropriate, as is the case with internal built in stored procedures. PHP query handling was modified to properly handle the case of stored procedures that do not return result sets.

6.10 Corrected Scale Value for c-treeACE SQL MONEY Types

c-treeACE SQL MONEY types should always have a fixed scale of 2. However, it was found that it was possible for this value to be set to 0. The default value of 2 is now correctly forced in all circumstances. An immediate field fix for existing databases that may exhibit this incorrect value is to execute an update statement on the system table:

```
UPDATE admin.syscolumns SET SCALE = 2 WHERE coltype = 'MONEY'
```

6.11 Avoid c-treeACE SQL Exception on Modified Imported Tables

c-treeACE SQL could unexpectedly terminate with an unhandled exception when reading a record from a table that was imported into a c-treeACE SQL database and whose *DODA* and SQL dictionary field definitions do not match. While not a normal occurrence, this condition could occur if a user modified the attributes of an imported table outside of c-treeACE SQL. An internal function allocated a buffer whose size is the maximum field length plus 1, however, it

passed the field length that was read from the *DODA* resource. If the *DODA* field length exceeds the allocated buffer size, it was possible to write past the end of the allocated buffer. To avoid this, if the allocated buffer length is smaller than the *DODA* field length, the full allocated buffer length is now passed.

6.12 c-treeACE SQL Default Value Usage Corrected for BIT Columns

An ALTER TABLE of the following construct:

```
ALTER TABLE groups ADD "IsExpired" bit not null default '1';
```

was failing with this error:

```
Error : -21042    Error Description : CTDB - Can't perform type conversion
```

BIT fields are mapped into c-treeACE type *CT_BOOL* and default values are set as strings. When adding the new field, the ALTER TABLE logic tried to apply the default value passing the string "1". However, *CT_BOOL* only converted the strings "true" and "false" (case insensitive).

Comparison logic was modified to allow proper Boolean checking.

6.13 Retrieval of LONG VARBINARY Fields with c-treeACE SQL ADO.NET Data Provider

It was found that the c-treeACE SQL ADO.NET Data Provider failed to return the final byte of a LONG VARBINARY field. An internal calculation on the length of the data in the field was found and corrected such that the complete field is now properly returned.

6.14 Reduced c-treeACE SQL ADO.NET Buffer Allocations Avoid .NET LOH

The previous c-treeACE SQL ADO.NET Provider default buffer allocation placed many objects into the .NET LOH (Large Object Heap). Microsoft Windows XP 32-bit and previous OS versions have a bug that prevents the LOH from being de-fragmented and can cause memory problems.

The default provider buffer allocations have been reduced to keep the object size below the 85KB limit (that caused objects to be placed in LOH) resulting in better usage of the .NET SOH (Small Objects Heap).

Important Considerations

- The modified allocations only reduce default object allocations. If a large number of rows is fetched or even few rows with large LVB or LVC columns, these objects will continue to be allocated in LOH due to .NET memory handling.
- According to information from Microsoft, the LOH de-fragmenting problem is a bug on older Windows versions. This has been corrected from Windows XP 64-bit onward. Should

memory problems or concerns continue, it is suggested to upgrade to a current version of Windows.

6.15 Improved LONG Column Support

An insert into a LONG column from a SELECT clause failed to update the value.

```
INSERT INTO IN_SCRIPT_TEMPTBL
(SCRIPT_ID,SCRIPT_TYPE,SCRIPT_NAME,SCRIPT_DESC,SCRIPT_LANGUAGE,CREATION_USR_ID,CREATION_TIME,
MOD_USR_ID,MOD_TIME,DATA)
SELECT
SCRIPT_ID,SCRIPT_TYPE,SCRIPT_NAME,SCRIPT_DESC,SCRIPT_LANGUAGE,CREATION_USR_ID,CREATION_TIME,M
OD_USR_ID,MOD_TIME,DATA FROM IN_SCRIPT;
```

LONG column handling has been reviewed, and several modifications were made to allow updates to these columns.

6.16 Query with Left Outer Join Returns Wrong Rows

A query had returned unexpected and incorrect results.

```
SELECT ip.INSTANCE_ID, ip.PROP_ID, p.PROP_NAME, v.VAL_NAME, v.prop_val_id, ip.STRING_VAL,
p.PROP_TYPE
FROM IN_INSTANCE i
LEFT OUTER JOIN IN_INSTANCE_PROP ip ON ip.INSTANCE_ID = i.INSTANCE_ID
LEFT OUTER JOIN IN_CLASS_PROP cp  ON i.CLASS_ID = cp.CLASS_ID AND ip.PROP_ID = cp.PROP_ID
INNER JOIN IN_PROP p ON p.PROP_ID = cp.PROP_ID
LEFT OUTER JOIN IN_PROP_VAL v ON v.prop_val_id = ip.STRING_VAL AND p.PROP_TYPE = 5
LEFT OUTER JOIN IN_USR u ON ip.STRING_VAL = u.USR_ID AND p.PROP_TYPE IN (6,7)
WHERE ip.INSTANCE_ID = '2000002CKX_0004FV4NX2EM'
ORDER BY cp.CLASS_ID, cp.SEQ_NUM;
```

Changes have been made to the query optimizer to prevent this behavior. Specifically, an internal condition was not properly detected and additional checks are now made to ensure a correct logic path is followed.

6.17 Correct SQL OR Clause Handling with Outer Joins and Nullable Columns

The following query returned no results from a data set that should have returned several rows:

```
SELECT id from qtable q
LEFT OUTER JOIN users u on q.id = u.usr_id
LEFT OUTER JOIN groups g on q.id = g.group_id
WHERE u.stat = 0 OR g.stat = 0;
```

The same query without the OR clause returned expected results.

An optimization within the OR operator logic contains a restrict node and based on the nullability of the restrict expressions, either a simple <> relational operator is used or a searched case. The nullability check for a field was based on whether the table column was nullable or not. However,

if the restrict was on a field that is from the right branch of an outer join, then it could be null even if the table column is not nullable.

There is now an added check if the join is an outer join and if the expression is from the RHS of the join the nullability is set accordingly.

6.18 Corrected VARCHAR String Handling for Empty Strings

It was found that an empty VARCHAR string could be considered a NULL value resulting in unexpected query results.

A recent enhancement to improve the performance of variable length string handling was found to inappropriately set empty VARCHAR fields to NULL when reading them from the file. This occurred only in recent V9.3 builds for tables created with the new VARCHAR mapping to a c-treeACE *CT_2STRING* type (the new default) and for imported tables using a *CT_PSTRING* type.

For tables having NULL field support ($NULFLD$, such as created trough c-treeACE SQL) a NULL value is cached earlier in the record handling and a NULL value is correctly returned. However, if the string was empty, a NULL value was incorrectly returned instead of the empty string. For tables without $NULFLD$ support, in theory there should be no NULL values at all, yet empty VARCHAR strings were also returned as NULL.

The logic was changed such that in cases for a VARCHAR field type and where the field buffer is null (indicating an empty string) an empty VARCHAR string is returned instead of the NULL value.

6.19 Proper Authorization When Using Table Alias

Queries from derived tables could throw an authorization error when a set schema was issued.

```
SET SCHEMA 'admin';

SELECT cm_custnum FROM (SELECT * FROM custmast) AS custno
error(-20228): Access denied(Authorisation failed)
```

While authorization was attempted on derived tables (aliased tables/query result), authorization information is not applied to these types of tables. The check is now skipped for these table types preventing this error.

6.20 64-bit c-treeACE SQL ODBC Driver Corrections

The 64-bit c-treeACE SQL ODBC Driver was found to not be fully functional or compliant. Various 64-bit porting issues regarding 64-bit memory addressing were identified and corrected.

6.21 c-treeACE SQL TOP Sort Optimization

A c-treeACE SQL query of the following form was found to run slow:

```
SELECT TOP N * FROM tbl WHERE tbl.a BETWEEN x AND y ORDER BY tbl.a, tbl.b;
```

When an index only exists on tbl.a, all of the records in the range are read and sorted, however, the TOP N value should be used to reduce this number, which can be a dramatic improvement when there are large numbers of records to be read. To optimize this query, once N records have been read, reading continues from tbl.a (already in order by index) only until tbl.a changes.

6.22 Improved Query Times with Multiple Operator Support in ANL Joins

A query was found to have its execution time out which contained the following clause:

```
(  table.field IN (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?) )
```

This IN clause was found to affect performance and it was determined that multiple operators were not allowed on index scans for an ANL join execution. Changes have been made to allow additional ANL predicates resulting in efficient index choices boosting performance.

6.23 c-treeACE SQL Query Timeout Now Resets to Zero

After setting a query timeout in ADO.NET (or any other client API), the timeout was applied to any query executed from that point on with no possibility of resetting it to 0 (infinite). A value is maintained indicating the timeout for the current operation (held connection wise). At the communication level, this value is changed for each call so that it stays current. Logic exists to calculate and update a maximum time whenever the operation requested sets a timeout different than 0. If the requested timeout was 0, this value was left unchanged, which may have caused unexpected timeouts. This was corrected by resetting the maximum to 0 when the timeout is 0.

6.24 c-treeACE SQL ADO.NET Provider Now Closes Thread for ThreadAbortException

The c-treeACE SQL ADO.NET provider has been modified to close the thread in the case of a *ThreadAbortException*, a somewhat non-standard situation pertaining to implicitly started transactions. Previously it would close the implicit transaction only for an *FcSqlException*. It is recommended that end user code should manually start transactions inside the thread and, in case the thread aborts, decide to rollback or commit. This change will ease the case of implicit transactions, where the user did not explicitly begin the transaction.

6.25 c-treeACE SQL ADO.NET ExecuteScalar() Altered to Close the Reader

A situation was found in which in the c-treeACE SQL ADO.NET provider's **ExecuteScalar()** method may not be calling the **reader.Close** function. This only occurred in certain circumstances

in which the reader itself caused an exception. The method has been altered to ensure the reader is always closed.

6.26 c-treeDB Internal Error Diagnostics

A **CTDBRET_INTERNAL** (4025) error was encountered when testing c-treeACE SQL. Extremely rare, this internal error indicates an unexpected condition for several locations within c-treeDB logic, extensively used by c-treeACE SQL. A new c-treeACE SQL configuration keyword was introduced to help locate the exact source of the error.

The configuration option `SQL_TRACE_CTREE_ERROR` *<error number>* allows the exact code location of errors to be narrowed and generates a process stack trace when a specific internal c-treeDB error is encountered. Should this error occur, this trace should be provided to FairCom to pinpoint the condition leading to this error.

Note: Depending upon where in the server an error is generated, It is possible for some errors to result in two stack traces being generated for certain errors.

6.27 c-treeACE SQL OR Optimization Enhancement

A slow query was identified that did not consider a possible optimization path. OR optimization was improved, ensuring that the execution node restriction was broken down even when there is only one table. Functionality was added to convert OR into UNION, which is beneficial if the cardinality of each ORed predicate is small.

A configuration keyword, `SETENV DH_OPT_OR_CARD`, was introduced to specify the threshold for this conversion. The default value is 10000.

6.28 Prevent Tuple ID Not Found Messages for Valid Result Sets

Testing showed an **ETPL_TUPLE_NOT_FOUND** (10002, Tuple not found for the specified TID) error was returned in a situation that was a valid result. This occurred when the query selected the TOP N results in a specified sort order and the result set contained fewer than N results as a result of DISTINCT removals, a valid result that should not return an error. The logic has been corrected so that this message is not returned under these conditions.

6.29 Corrected Unhandled Exception in Execution Plan Output

A server crash was reported when attempting to print an execution plan. When SQL_DEBUG ERROR_INFO is specified in *ctsrvr.cfg* and a SQL query fails, an attempt is made to print the

execution plan. An unchecked return code caused an unhandled exception if this function fails. This value is now always checked to ensure safe handling.

6.30 Corrected Unhandled Exception with c-treeACE SQL Connection Monitoring

A recent c-treeACE SQL connection monitoring enhancement introduced a condition in which c-treeACE SQL terminated with an unhandled exception when **USERINFO()** was called at the same time a client logged on. For example, **USERINFO()** is used in the c-treeACE Monitor tool for monitoring client connections. A connection state bit could potentially be null at logon, an acceptable case, however, was referenced without checking for this null condition. A check has been added to prevent this occurrence.

6.31 Corrected Memory Leak with DH_REBUILD_SEL_CUTOFF

Testing demonstrated a memory leak when using `DH_REBUILD_SEL_CUTOFF` SQL configuration option to force reoptimization of a parameterized query if the selectivity changes. The scope of the leak was the size of the statement at the time it was reoptimized. The logic has been changed to correctly free all memory.

6.32 Improvements for Certain Long-Running Queries

The type of query shown below was found to run slower than expected in certain situations:

```
SELECT (TOP k) * FROM tbl WHERE x BETWEEN a AND b ORDER BY y
```

The default cost estimates used in choosing an execution plan have been updated to improve general performance of this type of query. Additional options are also available to further adjust costing used in this optimization.

- `TPE_MM_COMPARE_OVHD` (default value is 1)
- `TPE_MM_INSERT_OVHD` (default value is 250)

6.33 Invalid String Error Resolved When Specifying Selectivity Option

Setting query parameters when the `DH_REBUILD_SEL_CUTOFF` configuration option is specified caused a query to fail with error SQL_ERR_INVNUMSTR. The parameter value assignment did

not correctly handle all parameter types before recalculating the selectivity of the query. All parameter types are now correctly processed with this option.

6.34 Optimize c-treeACE SQL Left Outer Joins to Inner Joins When Applicable

A query was reported that contained multiple outer joins ran very slowly for some values of query criteria. It was observed that the join conditions could allow some of the joins to be converted to inner joins, greatly reducing the number of records to be read. This optimization is now taken when the appropriate join conditions exist.

7. Replication Fixes

7.1 Correct Replication Agent Exception Handling with Failed File Opens

When the replication agent configuration file specifies the `exception_mode operation` option, if an add/delete/update operation in a transaction fails, other operations in the transaction are intended to be still be applied. (By contrast, the `exception_mode transaction` option aborts the transaction if any of the operations that comprise the transaction fail.)

It was found in testing that when applying operations for a transaction, the replication agent always aborted the transaction if applying an operation required a file to be opened and the attempt to open the file fails.

This modification changes the behavior of the replication agent, such that when `exception_mode operation` is in specified, if applying a change fails due to failing to open a file, the replication agent continues to attempt to apply the other operations for the transaction.

7.2 Lock Restoration when Local Record Reads Fail

When a local record read fails with error **TR_RDIF_ERR** (856, Transactional replication: Failed to read record for update: local record differs from master record), the local c-treeACE process now restores the lock state of the local and master records to their states prior to the record read operation.

To restore the master record lock state, the local server calls the **ctReplMasterControl()** function on the master server. This function is intended to support additional operations on the master server. At present, it accepts a *mode* parameter having the value *ctMASTERCTLrstlok* and a *data* parameter that is set to the data file number for the master file for which we want to restore the lock state of the record we just read.

If the call to **ctReplMasterControl()** fails, the local server writes one of the following messages to *CTSTATUS.FCS* and fails the record read operation with the error *<error_code>*:

```
ctTranReplGetFullRecord: Failed to unlock master record: <error_code>
ctTranReplReadMasterRecord: Failed to unlock master record: <error_code>
```

This behavior is designed such that it is easy to tell if a master Server does not support this function. In that case, the error code will likely be error **SFUN_ERR** (170, bad function number).

7.3 Corrected Client Hangs on Local Server Updates

A client call to a c-treeACE Server that is configured as a local server (that is connected to a master c-treeACE Server) could hang after updating or deleting a record on the local server.

A call to the local server to update or delete a record may need to re-read the full record from the local server, however, the call to re-read the record writes the record image to the communication

buffer, and the update or delete call then sends this record image to the client. But the client did not expect to receive a record image on an update or delete operation, and so the client became out of sync with the server with respect to the synchronous send and receive calls and would hang on a subsequent call to read a response from the server.

To avoid this out of sync communication, the communication buffer pointer is now saved and set to NULL before reading the record from the local server such such that the call to read the local record does not write to the communication buffer. After the call completes, the communication buffer pointer is then restored to its original value.

7.4 Prevent READ_ERR during Replication Log Reads

A **READ_ERR** (36) could be returned by the **ctReplGetNextChange()** API call, and a message such as the following found in *CTSTATUS.FCS*:

```
"Checksum failure at code location"
```

This error was possible when all of the following conditions were true:

- When reading the last entry of the transaction log
- The log entry header has been flushed, however, the two byte checksum had not yet been fully flushed.
- **ctReplGetNextChange()** specified a zero (*ctNOWAIT*) timeout.

When reading the log entry header no test of the checksum was made. When a timeout value was non-zero, a check was made that the checksum had been flushed, and if not, an **NTIM_ERR** was returned. When a timeout value was zero, no check was made that the checksum was on disk, the next call read the checksum, however, since it has not been flushed, it was 0xFF on disk. This caused the checksum test to fail and **READ_ERR** to be returned. A check is now always made at least once that the checksum is on disk, regardless of the timeout value preventing this error.

8. Tool Fixes

8.1 Corrected Display of LONG VARBINARY Fields in c-treeACE SQL Explorer

When viewing c-treeACE SQL LONG VARBINARY fields (c-tree *CT_4STRING*) in the SQL Explorer tool, incorrect values were displayed for some bytes. The *View As Generic Data* display has been corrected for these field types.

8.2 IF_EXIST Syntax Handling in c-treeACE SQL Explorer

It was found that the c-treeACE SQL Explorer didn't support the IF_EXIST and IF_NOT_EXIST syntax (not the underscore characters), however, it did support IF EXIST and IF NOT EXIST constructs. This additional support was added to the tool.

Note that this syntax is supported in "Load Script" mode (when the script is run in interactive mode), however, not yet supported in "Run Mode".

8.3 Corrected c-treeACE SQL ISQL Script Handling Parameters

The c-treeACE SQL ISQL utility was found to cause an exception when using the "@" command to specify an external script file. An internal buffer size was determined to not have enough room for the \0 nul string terminator. This buffer was increased by 1 to ensure no memory overwrites occur.

8.4 Updates to ctunf1 File Format Conversion Utility Corrected Byte Reversal of Extended Headers

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

Copyright Notice

Trademarks

11/16/2018

9. Index