Contents

1. Introduction ..............................................................................................................1
  1.1 Layout of this Manual ..........................................................................................2

2. Quick Tour .............................................................................................................3
  2.1 Introductory Tutorial ...........................................................................................4
    Init 5
    Define .....................................................................................................................6
    Manage ....................................................................................................................8
    Done .......................................................................................................................11
    Additional Resources .............................................................................................12
  2.2 Relationships ......................................................................................................13
    Init 15
    Define ....................................................................................................................16
    Manage ..................................................................................................................22
    Done .......................................................................................................................28
    Additional Resources .............................................................................................29
  2.3 Record/Row Locking ..........................................................................................30
    Init 32
    Define ....................................................................................................................33
    Manage ..................................................................................................................36
    Done .......................................................................................................................40
    Additional Resources .............................................................................................41
  2.4 Transaction Processing .....................................................................................42
    Init 44
    Define ....................................................................................................................45
    Manage ..................................................................................................................51
    Done .......................................................................................................................58
    Additional Resources .............................................................................................59

3. Programmer’s Reference .......................................................................................60
  3.1 Common functionality .......................................................................................61
    Error Handling ......................................................................................................61
    Transaction Processing .........................................................................................61
    Session Wide Locking .........................................................................................62
    Default Date, Time and Float formats ..................................................................63
    User Defined Tags ..................................................................................................64
  3.2 Working with Sessions .....................................................................................65
    Creating a Session object .....................................................................................65
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>76</td>
</tr>
<tr>
<td>3.3 Working with Databases</td>
<td>77</td>
</tr>
<tr>
<td>Creating a Database object</td>
<td>77</td>
</tr>
<tr>
<td>Connecting to a database</td>
<td>77</td>
</tr>
<tr>
<td>Database properties</td>
<td>78</td>
</tr>
<tr>
<td>Managing Tables</td>
<td>79</td>
</tr>
<tr>
<td>Next Table</td>
<td>81</td>
</tr>
<tr>
<td>Database Dictionary</td>
<td>84</td>
</tr>
<tr>
<td>3.4 Working with Tables</td>
<td>85</td>
</tr>
<tr>
<td>Creating a Table object</td>
<td>86</td>
</tr>
<tr>
<td>Creating a New Table</td>
<td>87</td>
</tr>
<tr>
<td>Opening a table</td>
<td>97</td>
</tr>
<tr>
<td>Altering a table</td>
<td>99</td>
</tr>
<tr>
<td>Attach and Detach Open Tables</td>
<td>106</td>
</tr>
<tr>
<td>Table Properties</td>
<td>107</td>
</tr>
<tr>
<td>3.5 Working with Records</td>
<td>113</td>
</tr>
<tr>
<td>Creating a record object</td>
<td>113</td>
</tr>
<tr>
<td>The default index</td>
<td>115</td>
</tr>
<tr>
<td>Navigating records</td>
<td>116</td>
</tr>
<tr>
<td>Finding records</td>
<td>118</td>
</tr>
<tr>
<td>Record sets</td>
<td>121</td>
</tr>
<tr>
<td>Record Filters</td>
<td>122</td>
</tr>
<tr>
<td>Record Batches</td>
<td>143</td>
</tr>
<tr>
<td>Reading and writing field data to a record buffer</td>
<td>153</td>
</tr>
<tr>
<td>Clearing the record buffer</td>
<td>157</td>
</tr>
<tr>
<td>Adding new records</td>
<td>158</td>
</tr>
<tr>
<td>Updating existing records</td>
<td>159</td>
</tr>
<tr>
<td>Deleting records</td>
<td>160</td>
</tr>
<tr>
<td>Record properties</td>
<td>160</td>
</tr>
<tr>
<td>Record Locking</td>
<td>161</td>
</tr>
<tr>
<td>3.6 Data Types</td>
<td>163</td>
</tr>
<tr>
<td>Scalar Types</td>
<td>163</td>
</tr>
<tr>
<td>Date Types</td>
<td>163</td>
</tr>
<tr>
<td>Time Types</td>
<td>165</td>
</tr>
<tr>
<td>Date/Time (Timestamp) Types</td>
<td>165</td>
</tr>
<tr>
<td>Numeric Types</td>
<td>166</td>
</tr>
</tbody>
</table>
3.7 Data Integrity ................................................................. 170
Transactions................................................................. 170
Locking................................................................. 172
3.8 Working with Resources ................................................................. 174
Types of Resources ................................................................. 175
c-treeDB C++ API - Working with Resources ................................................................. 176
3.9 Working with Callbacks ................................................................. 184
Callback Function Type ................................................................. 184
Callback Return Codes ................................................................. 185
Callback Handle Parameters ................................................................. 185
Callback Types ................................................................. 186
Working with Callbacks ................................................................. 205
3.10 Working with Unicode ................................................................. 207
c-treeDB Support for Unicode ................................................................. 207
Unicode UTF-16 ................................................................. 207
Unicode UTF-8 ................................................................. 208
c-treeDB C++ API UTF-8 Compliance ................................................................. 208
Activating c-treeDB Unicode support ................................................................. 209
ICU - International Components for Unicode ................................................................. 209
Unicode Support ................................................................. 209
Extended Key Segment Definition ................................................................. 216
3.11 Compatibility with other c-tree API Technologies ................................................................. 220
Compatibility with c-treeACE ISAM and Low-Level Data Files ................................................................. 220
Compatibility with c-treeACE SQL ................................................................. 220
3.12 c-treeDB, ISAM, and Low-Level Integration ................................................................. 222
Overview ................................................................. 222
Switching c-tree instances ................................................................. 222
Switching ISAM contexts ................................................................. 223
Obtaining table data and file number ................................................................. 224
Obtaining data file number ................................................................. 224
Obtaining index file number ................................................................. 225
4. c-treeDB C++ API Class Reference ................................................................. 226
4.1 c-treeDB Definitions ................................................................. 227
Field Types ................................................................. 227
Find Modes ................................................................. 230
Index Key Types ................................................................. 231
Record Lock Modes ................................................................. 231
Session Wide Lock Modes ................................................................. 231
Segment Modes ................................................................. 232
Table Create Modes ................................................................. 234
Table Open Modes ................................................................. 235
# Table Permissions

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>CTBase Class</td>
<td>238</td>
</tr>
<tr>
<td>class CTBase</td>
<td></td>
<td>238</td>
</tr>
<tr>
<td>CTPublic Members</td>
<td></td>
<td>238</td>
</tr>
<tr>
<td>CTPublic Methods</td>
<td></td>
<td>239</td>
</tr>
<tr>
<td>4.3</td>
<td>CTBigint Class</td>
<td>282</td>
</tr>
<tr>
<td>class CTBigint</td>
<td></td>
<td>282</td>
</tr>
<tr>
<td>CTBigint Operators</td>
<td></td>
<td>282</td>
</tr>
<tr>
<td>CTBigint Methods</td>
<td></td>
<td>285</td>
</tr>
<tr>
<td>4.4</td>
<td>CTBlob Class</td>
<td>296</td>
</tr>
<tr>
<td>class CTBlob</td>
<td></td>
<td>296</td>
</tr>
<tr>
<td>CTBlob Operators</td>
<td></td>
<td>296</td>
</tr>
<tr>
<td>CTBlob Methods</td>
<td></td>
<td>299</td>
</tr>
<tr>
<td>4.5</td>
<td>CTCurrency Class</td>
<td>309</td>
</tr>
<tr>
<td>class CTCurrency</td>
<td></td>
<td>309</td>
</tr>
<tr>
<td>CTCurrency Operators</td>
<td></td>
<td>309</td>
</tr>
<tr>
<td>CTCurrency Methods</td>
<td></td>
<td>312</td>
</tr>
<tr>
<td>4.6</td>
<td>CTDdatabase Class</td>
<td>326</td>
</tr>
<tr>
<td>class CTDdatabase</td>
<td></td>
<td>326</td>
</tr>
<tr>
<td>CTDdatabase Methods</td>
<td></td>
<td>327</td>
</tr>
<tr>
<td>4.7</td>
<td>CTDDate Class</td>
<td>351</td>
</tr>
<tr>
<td>class CTDDate</td>
<td></td>
<td>351</td>
</tr>
<tr>
<td>CTDDate Operators</td>
<td></td>
<td>351</td>
</tr>
<tr>
<td>CTDDate Methods</td>
<td></td>
<td>354</td>
</tr>
<tr>
<td>4.8</td>
<td>CTDDateTime Class</td>
<td>369</td>
</tr>
<tr>
<td>class CTDDateTime</td>
<td></td>
<td>369</td>
</tr>
<tr>
<td>CTDDateTime Operators</td>
<td></td>
<td>369</td>
</tr>
<tr>
<td>CTDDateTime Methods</td>
<td></td>
<td>372</td>
</tr>
<tr>
<td>4.9</td>
<td>CTException Class</td>
<td>393</td>
</tr>
<tr>
<td>class CTException</td>
<td></td>
<td>393</td>
</tr>
<tr>
<td>CTException Methods</td>
<td></td>
<td>394</td>
</tr>
<tr>
<td>4.10</td>
<td>CTFIELD Class</td>
<td>401</td>
</tr>
<tr>
<td>class CTFIELD</td>
<td></td>
<td>401</td>
</tr>
<tr>
<td>CTFIELD Operators</td>
<td></td>
<td>401</td>
</tr>
<tr>
<td>CTFIELD Methods</td>
<td></td>
<td>403</td>
</tr>
<tr>
<td>4.11</td>
<td>CINDEX Class</td>
<td>430</td>
</tr>
<tr>
<td>class CINDEX</td>
<td></td>
<td>430</td>
</tr>
<tr>
<td>CINDEX Operators</td>
<td></td>
<td>430</td>
</tr>
<tr>
<td>CINDEX Methods</td>
<td></td>
<td>432</td>
</tr>
<tr>
<td>4.12</td>
<td>CTMONEY Class</td>
<td>462</td>
</tr>
<tr>
<td>class CTMONEY</td>
<td></td>
<td>462</td>
</tr>
</tbody>
</table>
1. Introduction

FairCom is pleased to present the c-tree database API: c-treeDB. This API gives application developers a simple interface into the powerful core of c-tree, yet includes the advanced functionality that distinguishes c-tree from other database solutions. c-treeDB offers a method of database creation and maintenance that is easier to use than the traditional c-tree ISAM and low-level APIs. The c-tree database utilizes the simplified concepts of sessions, databases, and tables in addition to the standard concepts of records, fields, indices, and segments. This new database API allows for effortless and productive management of database systems.

Overview

The c-treeDB C++ API provides libraries that, once installed and correctly referenced, allow C++ programs to access the c-tree database core via function calls. This approach provides application developers with a simple interface into the powerful core of c-tree, yet includes the advanced functionality that distinguishes c-tree from other database solutions. c-treeDB offers a method of database creation and maintenance that is easier to use than the traditional c-treeACE ISAM and c-treeACE Low-Level APIs.
c-treeDB utilizes the simplified concepts of sessions, databases, and tables in addition to the standard concepts of records, fields, indices, and segments. This database layer allows for effortless and productive management of database systems.

**c-treeDB Relationships**

![Diagram of c-treeDB relationships](image)

c-treeDB consists of two separate APIs. The C++ API provides the classes and methods that comprise the database functionality. The C API provides the function calls without an object-oriented schema. Aside from the schematic differences, there are few distinctions between these C++ and C c-treeDB APIs.

### 1.1 Layout of this Manual

This manual, which describes the C++ API, is divided into the following major chapters:

**Chapter 2 - Quick Tour**

The quick tour section will show how easy it is to develop database applications using the c-treeDB C++ API. Basically you should initialize, define, manage, and you are done. Please take a look at our introductory tutorials for a quick introduction into c-treeDB database programming.

**Chapter 3 - Programmer’s Reference**

The programmer’s reference presents a detailed description of the concepts exported by c-treeDB API. Each concept is presented with a detailed description and programming examples.

**Chapter 4 - Class Reference**

The class reference details the classes that comprise the c-treeDB C++ API.
2. Quick Tour

This chapter will introduce four quick tutorials showing the major components of our core technology:

- How to init, define and manage data
- How to handle index files
- How to use locks in a multi-user environment
- How to use transaction processing
2.1 Introductory Tutorial

This tutorial will take you through the basic use of the c-treeACE C++ Database Framework.

Like all other examples in the c-tree tutorial series, this tutorial simplifies the creation and use of a database into four simple steps: Initialize(), Define(), Manage(), and You're Done()!

Tutorial #1: Introductory - Simple Single Table

We wanted to keep this program as simple as possible. This program does the following:

- Initialize() - Connects to the c-treeACE Database Engine.
- Define() - Defines and creates a "customer master" (custmast) table/file.
- Manage() - Adds a few rows/records; Reads the rows/records back from the database; displays the column/field content; and then deletes the rows/records.
- Done() - Disconnects from c-treeACE Database Engine.

Note our simple mainline:

```c
//
// main()
// The main() function implements the concept of "init, define, manage
// and you're done..."
//
int main(int argc, pTEXT argv[])
{
    Initialize();
    Define();
    Manage();
    Done();
    printf("\nPress <ENTER> key to exit . . .\n");
    getchar();
    return(0);
}
```

We suggest opening the source code with your own editor.

Continue now to review these four steps.
First we need to open a connection to a database by providing the c-treeACE Database Engine with a user name, password and the database name.

Below is the code for Initialize():

```c
// Initialize()
// Perform the minimum requirement of logging onto the c-tree Server

VOID Initialize(VOID)
{
    printf("INIT\n");
    try
    {
        // connect to server
        printf("\rLogon to server...
");
        MySession.Logon("FAIRCOMS", ", ", ");
    }
    catch(CTException E)
    {
        Handle_Exception(E);
    }
}
```
Define

The define step is where specific data definitions are established by your application and/or process. This involves defining columns/fields and creating the tables/files with optional indices.

Below is the code for Define():

```c++
//
// Define()
//
// Open the table, if it exists. Otherwise create and open the table
//
VOID Define(VOID)
{
    CTBOOL do_create = NO;

    printf("DEFINE\n");
    try
    {
        printf("\tOpen table...\n");
        MyTable.Open("custmast", CTOPEN_NORMAL);
    } catch (...)
    {
        // table does not exist. Try to create it
        do_create = YES;
    }

    if (do_create)
    {
        // create the table
        printf("\tAdd fields...\n");
        try
        {
            MyTable.AddField("cm_custnumb", CT_FSTRING, 4);
            MyTable.AddField("cm_custzipc", CT_FSTRING, 9);
            MyTable.AddField("cm_custstat", CT_FSTRING, 2);
            MyTable.AddField("cm_custrtng", CT_FSTRING, 1);
            MyTable.AddField("cm_custname", CT_STRING, 47);
            MyTable.AddField("cm_custaddr", CT_STRING, 47);
            MyTable.AddField("cm_custcity", CT_STRING, 47);

            printf("\tCreate table...\n");
            MyTable.Create("custmast", CTCREATE_NORMAL);
            MyTable.Open("custmast", CTCREATE_NORMAL);
        }
        catch (CTException E)
        {
            Handle_Exception(E);
        }
    }
    else
```
VOID Check_Table_Mode(CTTable& table)
{
    try
    {
        // get table create mode
        CTCREATE_MODE mode = table.GetCreateMode();

        // check if table is under transaction processing control
        if ((mode & CTCREATE_TRNLOG))
        {
            // change file mode to disable transaction processing
            mode ^= CTCREATE_TRNLOG;
            table.UpdateCreateMode(mode);
        }
    }
    catch (CTException E)
    {
        Handle_Exception(E);
    }
}
Manage

The manage step provides data management functionality for your application and/or process.

Below is the code for Manage():

```c
// Manage()
// This function performs simple record functions of add, delete and gets

VOID Manage(VOID)
{
    printf("MANAGE\n");

    // delete any existing records
    Delete_Records();

    // populate the table with data
    Add_Records();

    // display contents of table
    Display_Records();
}

// Delete_Records()
// This function deletes all the records in the table

VOID Delete_Records(VOID)
{
    CTBOOL found;

    printf("\tDelete records...\n");

    try
    {
        // read first record
        found = MyRecord.First();

        while (found) // while records are found
        {
            // delete record
            MyRecord.Delete();
            // read next record
            found = MyRecord.Next();
        }
    }
    catch(CTException E)
    {
        Handle_Exception(E);
    }
```
VOID Add_Records(VOID)
{
    typedef struct {
        cpTEXT number, zipcode, state, rating, name, address, city;
    } DATA_RECORD;

    DATA_RECORD data[] = {
        "1000", "92867", "CA", "1", "Bryan Williams", "2999 Regency", "Orange",
        "1001", "61434", "CT", "1", "Michael Jordan", "13 Main", "Harford",
        "1003", "10034", "MO", "1", "Keyon Dooling", "19771 Park Avenue", "Columbia"
    };
    CTSIGNED nRecords = sizeof(data) / sizeof(DATA_RECORD);

    try
    {
        for(CTSIGNED i = 0; i < nRecords; i++)
        {
            MyRecord.Clear();
            // populate record buffer with data
            MyRecord.SetFieldAsString(0, data[i].number);
            MyRecord.SetFieldAsString(1, data[i].zipcode);
            MyRecord.SetFieldAsString(2, data[i].state);
            MyRecord.SetFieldAsString(3, data[i].rating);
            MyRecord.SetFieldAsString(4, data[i].name);
            MyRecord.SetFieldAsString(5, data[i].address);
            MyRecord.SetFieldAsString(6, data[i].city);

            // add record
            MyRecord.Write();
        }
    }
    catch(CTException E)
    {
        Handle_Exception(E);
    }
}

VOID Display_Records(VOID)
{
    CTBOOL found;
}
TEXT custnumb[47+1];
TEXT custname[47+1];

printf("\tDisplay records...");

try {
  // read first record
  found = MyRecord.First();

  while (found) {
    strcpy(custnumb, MyRecord.GetFieldAsString(0).c_str());
    strcpy(custname, MyRecord.GetFieldAsString(4).c_str());

    printf("\n\t\t%8s%20s\n", custnumb, custname);

    // read next record
    found = MyRecord.Next();
  }
  catch (CTException E) {
    Handle_Exception(E);
  }
}
Done

When an application and/or process has completed operations with the database, it must release resources by disconnecting from the database engine.

Below is the code for **Done()**:

```c
//
// Done()
//
// This function handles the housekeeping of closing, freeing,
// disconnecting and logging out of the database
//

VOID Done(VOID)
{
    printf("DONE\n");

    try
    {
        // close table
        printf("\tClose table...\n");
        MyTable.Close();
        // logout
        printf("\tLogout...\n");
        MySession.Logout();
    }
    catch(CTException E)
    {
        Handle_Exception(E);
    }
} 
```
Additional Resources

We encourage you to explore the additional resources listed here:

- Complete source code for this tutorial can be found in ctpp_tutorial1.cpp in your installation directory, within the 'sdk\ctree.cpp\tutorials' directory for your platform. Example for the Windows platform: C:\FairCom\V*\win32\sdk\ctree.cpp\tutorials\ctpp_tutorial1.cpp (where C:\FairCom\V* is your installation directory).

- Additional documentation may be found on the FairCom Web site at: www.faircom.com
2.2 Relationships

Now we will build some table/file relationships using the c-treeACE C++ Database Framework.

This tutorial will advance the concepts introduced in the first tutorial by expanding the number of tables. We will define key columns/fields and create specific indices for each table to form a relational model database.

Like all other examples in the c-tree tutorial series, this tutorial simplifies the creation and use of a database into four simple steps: Initialize(), Define(), Manage(), and You’re Done()!

Tutorial #2: Relational Model and Indexing

Here we add a bit more complexity, introducing multiple tables, with related indices in order to form a simple “relational” database simulating an Order Entry system. Here is an overview of what will be created:

**Relational Model Tables**

- **Initialize()** - Connects to the c-treeACE Database Engine.
- **Define()** - Defines and creates the "custmast", "custordr", "ordritem" and the "itemmast" tables/files with related indices.
- **Manage()** - Adds some related rows/records to all tables/files. Then queries the database.
- **Done()** - Disconnects from c-treeACE Database Engine.

Note our simple mainline:

```c
//
// main()
//
// The main() function implements the concept of "init, define, manage
// and you're done..."
//
int main (COUNT argc, pTEXT argv[])
{
```
Quick Tour

Initialize();
Define();
Manage();
Done();

printf("\nPress <ENTER> key to exit . . .\n");
getchar();
return(0);
}

We suggest opening the source code with your own editor.

Continue now to review these four steps.
First we need to open a connection to a database by providing the c-treeACE Database Engine with a user name, password and the database name.

Below is the code for Initialize():

```c
// Initialize()
// Perform the minimum requirement of logging onto the c-tree Server
//
VOID Initialize(VOID)
{
    printf("INIT\n");
    try
    {
        // connect to server
        printf("\tLogon to server...\n");
        MySession.Logon("FAIRCOMS", ", ");
    }
    catch(CTException E)
    {
        Handle_Exception(E);
    }
```
Define

The define step is where specific data definitions are established by your application and/or process. This involves defining columns/fields and creating the tables/files with optional indices.

Below is the code for Define():

```c
// Define()

VOID Define(VOID)
{
    printf("DEFINE\n");
    Create_CustomerMaster_Table();
    Create_CustomerOrders_Table();
    Create_OrderItems_Table();
    Create_ItemMaster_Table();
}

//
// Create_CustomerMaster_Table()

VOID Create_CustomerMaster_Table(VOID)
{
    CTBOOL do_create = NO;
    // define table CustomerMaster
    printf("\table CustomerMaster\n");
    try
    {
        tableCustMast.Open("custmast", CTOPEN_NORMAL);
    }
    catch (...)
    {
        // table does not exist
        do_create = YES;
    }
    if (do_create)
    {
        try
        {
            // define table fields
            CTFIELD field1 = tableCustMast.AddField("cm_custnumb", CT_FSTRING, 4);
            tableCustMast.AddField("cm_custzipc", CT_FSTRING, 9);
        }
```
tableCustMast.AddField("cm_custstat", CT_FSTRING, 2);
tableCustMast.AddField("cm_custratg", CT_FSTRING, 1);
tableCustMast.AddField("cm_custname", CT_STRING, 47);
tableCustMast.AddField("cm_custaddr", CT_STRING, 47);
tableCustMast.AddField("cm_custcity", CT_STRING, 47);

// define index
CTIndex index1 = tableCustMast.AddIndex("cm_custnumb_idx", CTINDEX_FIXED, NO, NO);
index1.AddSegment(field1, CTSEG_SCHSEG);

// create table
printf("\tCreate table...
");
tableCustMast.Create("custmast", CTCREATE_NORMAL);

// open table
printf("\tOpen table...
");
tableCustMast.Open("custmast", CTOPEN_NORMAL);
}
catch (CTException E)
{
    Handle_Exception(E);
}

else
{
    Check_Table_Mode(tableCustMast);

    // confirm the index exists, if not then add the index
    //
    // this scenario arises out of the fact that this table was created in tutorial 1
    // without indices. The index is now created by the call to ctdbAlterTable

do_create = NO;
try
{
    tableCustMast.GetIndex("cm_custnumb_idx");
}
catch (CTException E)
{
    do_create = YES;
}

if (do_create)
{
try
{
    CTFIELD field1 = tableCustMast.GetField("cm_custnumb");
    CTIndex index1 = tableCustMast.AddIndex("cm_custnumb_idx", CTINDEX_FIXED, NO, NO);
    index1.AddSegment(field1, CTSEG_SCHSEG);
    tableCustMast.Alter(CTDB_ALTER_NORMAL);
}
catch (CTException E)
{
    Handle_Exception(E);
}
}
}
VOID Create_CustomerOrders_Table(VOID)
{
    CTBOOL do_create = NO;

    // Define table CustomerOrders
    printf("\ttable CustomerOrders\n");
    try
    {
        tableCustOrdr.Open("custordr", CTOPEN_NORMAL);
    }
    catch (...)
    {
        // Table does not exist
        do_create = YES;
    }

    if (do_create)
    {
        try
        {
            // Define table fields
            tableCustOrdr.AddField("co_ordrdate", CT_DATE, 4);
            tableCustOrdr.AddField("co_promdate", CT_DATE, 4);
            CTField field1 = tableCustOrdr.AddField("co_ordrnumb", CT_FSTRING, 6);
            CTField field2 = tableCustOrdr.AddField("co_custnumb", CT_FSTRING, 4);

            // Define indices
            CTIndex index1 = tableCustOrdr.AddIndex("co_ordrnumb_idx", CTINDEX_LEADING, NO, NO);
            index1.AddSegment(field1, CTSEG_SCHSEG);
            CTIndex index2 = tableCustOrdr.AddIndex("co_custnumb_idx", CTINDEX_LEADING, YES, NO);
            index2.AddSegment(field2, CTSEG_SCHSEG);

            // Create table
            printf("\tCreate table...\n");
            tableCustOrdr.Create("custordr", CTCREATE_NORMAL);

            // Open table
            printf("\tOpen table...\n");
            tableCustOrdr.Open("custordr", CTOPEN_NORMAL);
        }
        catch (CTException E)
        {
            Handle_Exception(E);
        }
        else
        {
            Check_Table_Mode(tableCustOrdr);
        }
    }
}

// Create_OrderItems_Table()
// Open table OrderItems, if it exists. Otherwise create it
// along with its indices and open it
VOID Create_OrderItems_Table(VOID)
{
    CTBOOL    do_create = NO;

    // define table OrderItems
    printf("\table OrderItems\n");
    try
    {
        tableOrdrItem.Open("ordritem", CTOPEN_NORMAL);
    }
    catch (...)
    {
        // table does not exist
        do_create = YES;
    }

    if (do_create)
    {
        try
        {
            // define table fields
            CTField field1 = tableOrdrItem.AddField("oi_sequnumb", CT_INT2, 2);
            tableOrdrItem.AddField("oi_quantity", CT_INT2, 2);
            CTField field2 = tableOrdrItem.AddField("oi_ordrnumb", CT_FSTRING, 6);
            CTField field3 = tableOrdrItem.AddField("oi_itemnumb", CT_FSTRING, 5);

            // define indices
            CTIndex index1 = tableOrdrItem.AddIndex("oi_ordrnumb_idx", CTINDEX_LEADING, NO, NO);
            index1.AddSegment(field2, CTSEG_SCHSEG);
            index1.AddSegment(field1, CTSEG_SCHSEG);
            CTIndex index2 = tableOrdrItem.AddIndex("oi_itemnumb_idx", CTINDEX_LEADING, YES, NO);
            index2.AddSegment(field3, CTSEG_SCHSEG);

            // create table
            printf("\tCreate table...
");
            tableOrdrItem.Create("ordritem", CTCREATE_NORMAL);

            // open table
            printf("\tOpen table...
");
            tableOrdrItem.Open("ordritem", CTOPEN_NORMAL);
        }
        catch (CTException E)
        {
            Handle_Exception(E);
        }
    }
    else
    
        Check_Table_Mode(tableOrdrItem);
    }

    // Create_ItemMaster_Table()
    //
    // Open table ItemMaster, if it exists. Otherwise create it
    // along with its indices and open it
    //
    VOID Create_ItemMaster_Table(VOID)
CTBOOL do_create = NO;

// define table ItemMaster
printf("\ttable ItemMaster\n");
try
{
    tableItemMast.Open("itemmast", CTOpen_NORMAL);
} catch (...)
{
    // table does not exist
    do_create = YES;
}
if (do_create)
{
    try
    {
        // define table fields
        tableItemMast.AddField("im_itemwght", CT_INT4, 4);
        tableItemMast.AddField("im_itempric", CT_MONEY, 4);
        CTField field1 = tableItemMast.AddField("im_itemnumb", CT_FSTRING, 5);
        tableItemMast.AddField("im_itemdesc", CT_STRING, 47);

        // define indices
        CTIndex index1 = tableItemMast.AddIndex("im_itemnumb_idx", CTINDEX.Fixed, NO, NO);
        index1.AddSegment(field1, CTSEG_SCHSEG);

        // create table
        printf("\tCreate table...
");
        tableItemMast.Create("itemmast", CTCREATE.NORMAL);

        // open table
        printf("\tOpen table...
");
        tableItemMast.Open("itemmast", CTOpen.NORMAL);
    } catch (CTException E)
    {
        Handle_Exception(E);
    }
    else
        Check_Table_Mode(tableItemMast);
}

// Check_Table_Mode()

// Check if existing table has transaction processing flag enabled.
// If a table is under transaction processing control, modify the
// table mode to disable transaction processing
//
VOID Check_Table_Mode(CTTable& table)
{
    try
    {
        // get table create mode
        CTCREATE_MODE mode = table.GetCreateMode();

// check if table is under transaction processing control
if ((mode & CCREATE_TRNLOG))
{
    // change file mode to disable transaction processing
    mode ^= CCREATE_TRNLOG;
    table.UpdateCreateMode(mode);
}

} catch (CTException E)
{
    Handle_Exception(E);
}
Manage

The manage step provides data management functionality for your application and/or process. Below is the code for `Manage()`:

```c
// Manage()
// Populates table and perform a simple query

VOID Manage(VOID)
{
  CTSIGNED quantity;
  CTFLOAT  price, total;
  TEXT     itemnumb[5+1], custnumb[4+1], ordrnumb[6+1], custname[47+1];
  CTBOOL   isOrderFound, isItemFound;

  printf("MANAGE\n");

  // populate the tables with data
  Add_CustomerMaster_Records();
  Add_CustomerOrders_Records();
  Add_OrderItems_Records();
  Add_ItemMaster_Records();

  // perform a query:
  // list customer name and total amount per order
  // name               total
  // @@@@@@@@@@@@@      $xx.xx
  // for each order in the CustomerOrders table
  //   fetch order number
  //   fetch customer number
  //   fetch name from CustomerMaster table based on customer number
  //   for each order item in OrderItems table
  //     fetch item quantity
  //     fetch item number
  //     fetch item price from ItemMaster table based on item number
  //   next
  // next

  try
  {
    // get the first order
    isOrderFound = recordCustOrdr.First();

    while (isOrderFound) // for each order in the CustomerOrders table
    {
      // fetch order number
```
strcpy(ordrnumb, recordCustOrdr.GetFieldAsString(2).c_str());
// fetch order number
strcpy(custnumb, recordCustOrdr.GetFieldAsString(3).c_str());
// fetch name from CustomerMaster table based on customer number
recordCustMast.Clear();
recordCustMast.SetFieldAsString(0, custnumb);
if (!recordCustMast.Find(CTFIND_EQ))
    continue;   // not possible in our canned example
strcpy(custname, recordCustMast.GetFieldAsString(4).c_str());
// fetch item price from OrderItems table
recordOrdrItem.Clear();
recordOrdrItem.SetFieldAsString(2, ordrnumb);
// define a recordset to scan only items applicable to this order
recordOrdrItem.RecordSetOn(6);
isItemFound = recordOrdrItem.First();

total = 0;
while (isItemFound) // for each order item in OrderItems table
{
    // fetch item quantity
    quantity = recordOrdrItem.GetFieldAsSigned(1);
    // fetch item number
    strcpy(itemnumb, recordOrdrItem.GetFieldAsString(3).c_str());

    // fetch item price from ItemMaster table based on item number
    recordItemMast.Clear();
    recordItemMast.SetFieldAsString(2, itemnumb);
    recordItemMast.Find(CTFIND_EQ);
    price = recordItemMast.GetFieldAsFloat(1);

    // calculate order total
    total += (price * quantity);

    isItemFound = recordOrdrItem.Next();
}

recordOrdrItem.RecordSetOff();
// output data to stdout
printf("\t\t-20s %.2f\n", custname, total);
// read next order
if (!recordCustOrdr.Next())
    isOrderFound = 0;
}
catch(CTException E)
{
    Handle_Exception(E);
}
}
VOID Add_CustomerMaster_Records(VOID)
{
    typedef struct {
        cpTEXT number, zipcode, state, rating, name, address, city;
    } DATA_RECORD;

    DATA_RECORD data[] = {
        "1000", "92867", "CA", "1", "Bryan Williams", "2999 Regency", "Orange",
        "1001", "61434", "CT", "1", "Michael Jordan", "13 Main", "Harford",
        "1003", "10034", "MO", "1", "Keyon Dooling", "19771 Park Avenue", "Columbia"
    };

    CTSIGNED nRecords = sizeof(data) / sizeof(DATA_RECORD);

    Delete_Records(recordCustMast);

    printf("\tAdd records in table CustomerMaster...\n");

    try
    {
        for (CTSIGN i = 0; i < nRecords; i++)
        {
            recordCustMast.Clear();

            // populate record buffer with data
            recordCustMast.SetFieldAsString(0, data[i].number);
            recordCustMast.SetFieldAsString(1, data[i].zipcode);
            recordCustMast.SetFieldAsString(2, data[i].state);
            recordCustMast.SetFieldAsString(3, data[i].rating);
            recordCustMast.SetFieldAsString(4, data[i].name);
            recordCustMast.SetFieldAsString(5, data[i].address);
            recordCustMast.SetFieldAsString(6, data[i].city);

            // add record
            recordCustMast.Write();
        }
    }
    catch(CTException E)
    {
        Handle_Exception(E);
    }
}

// Add_CustomerOrders_Records()
// This function adds records to table CustomerOrders from an array of strings

VOID Add_CustomerOrders_Records(VOID)
{
    CDate orderdate;
    CDate promisedate;

    typedef struct {
        cpTEXT orderdate, promisedate, ordernum, custnum;
    } DATA_RECORD;
DATA_RECORD data[] = {
    {"09/01/2002", "09/05/2002", "1", "1001"},
    {"09/02/2002", "09/06/2002", "2", "1002"}
};

CTSIGNEED nRecords = sizeof(data) / sizeof(DATA_RECORD);

Delete_Records(recordCustOrdr);

printf("\nAdd records in table CustomerOrders...\n");

try {
    for (CTSIGNEED i = 0; i < nRecords; i++) {
        recordCustOrdr.Clear();
        orderdate.StringToData(data[i].orderdate, CTDATE_MDCY);
        promisedate.StringToData(data[i].promisedate, CTDATE_MDCY);
        // populate record buffer with data
        recordCustOrdr.SetFieldAsDate(0, orderdate);
        recordCustOrdr.SetFieldAsDate(1, promisedate);
        recordCustOrdr.SetFieldAsString(2, data[i].ordernum);
        recordCustOrdr.SetFieldAsString(3, data[i].customernum);
        // add record
        recordCustOrdr.Write();
    }
}

catch(CTException E) {
    Handle_Exception(E);
}

//
// Add_OrderItems_Records()
//
// This function adds records to table OrderItems from an
// array of strings
//
VOID Add_OrderItems_Records(VOID)
{
    typedef struct {
        CTSIGNED sequencenum, quantity;
        cpTEXT ordernum, itemnum;
    } DATA_RECORD;

    DATA_RECORD data[] = {
        {1, 2, "1", "1"},
        {2, 1, "1", "2"},
        {3, 1, "1", "3"},
        {1, 3, "2", "3"}
    };
    CTSIGNED nRecords = sizeof(data) / sizeof(DATA_RECORD);

    Delete_Records(recordOrdrItem);
printf("Add records in table OrderItems...\n");

try {
    for (CTSIGNED i = 0; i < nRecords; i++) {
        recordOrdrItem.Clear();

        // populate record buffer with data
        recordOrdrItem.SetFieldAsSigned(0, data[i].sequencenum);
        recordOrdrItem.SetFieldAsSigned(1, data[i].quantity);
        recordOrdrItem.SetFieldAsString(2, data[i].ordernum);
        recordOrdrItem.SetFieldAsString(3, data[i].itemnum);

        // add record
        recordOrdrItem.Write();
    }
}

} catch (CTException E) {
    Handle_Exception(E);
}

// Add_ItemMaster_Records()
// This function adds records to table ItemMaster from an array of strings

VOID Add_ItemMaster_Records(VOID)
{
    typedef struct {
        CTSIGNED weight;
        CTMONEY  price;
        cpTEXT itemnum, description;
    } DATA_RECORD;

    DATA_RECORD data[] = {
        {10, 1995, "1", "Hammer"},
        {3, 999, "2", "Wrench"},
        {4, 1659, "3", "Saw"},
        {1, 398, "4", "Pliers"}
    };
    CTSIGNED nRecords = sizeof(data) / sizeof(DATA_RECORD);

    Delete_Records(recordItemMast);
    printf("Add records in table ItemMaster...\n");

    try {
        for (CTSIGNED i = 0; i < nRecords; i++) {
            recordItemMast.Clear();

            // populate record buffer with data
            recordItemMast.SetFieldAsSigned(0, data[i].weight);
            recordItemMast.SetFieldAsSigned(1, data[i].price);
            recordItemMast.SetFieldAsString(2, data[i].itemnum);
            recordItemMast.SetFieldAsString(3, data[i].description);

            // add record
            recordItemMast.Write();
        }
    }
}
// populate record buffer with data
recordItemMast.SetFieldAsSigned(0, data[i].weight);
recordItemMast.SetFieldAsMoney(1, data[i].price);
recordItemMast.SetFieldAsString(2, data[i].itemnum);
recordItemMast.SetFieldAsString(3, data[i].description);

    // add record
    recordItemMast.Write();

}
}

catch(CTException E)
{
    Handle_Exception(E);
}
}

// Delete_Records()
// This function deletes all the records in the table
//
VOID Delete_Records(CTRecord& record)
{
    CTBOOL   found;

    printf("\tDelete records...\n");

    try
    {
        // read first record
        found = record.First();

        while (found)  // while records are found
        {
            // delete record
            record.Delete();
            // read next record
            found = record.Next();
        }
    }
    catch(CTException E)
    {
        Handle_Exception(E);
    }
}
When an application and/or process has completed operations with the database, it must release resources by disconnecting from the database engine.

Below is the code for `Done()`:

```c
//
// Done()
//
// This function handles the housekeeping of closing, freeing,
// disconnecting and logging out of the database
//
VOID Done(VOID)
{
    printf("DONE\n");
    // close table
    printf("\tClose tables...\n");
    try
    {
        tableCustMast.Close();
        tableCustOrdr.Close();
        tableOrdrItem.Close();
        tableItemMast.Close();

        MySession.Logout();
    }
    catch(CTException E)
    {
        Handle_Exception(E);
    }
}
```
Additional Resources

We encourage you to explore the additional resources listed here:

- Complete source code for this tutorial can be found in ctpp_tutorial2.cpp in your installation directory, within the 'sdk\ctree.cpp\tutorials' directory for your platform. Example for the Windows platform: C:\FairCom\V*\win32\sdk\ctree.cpp\tutorials\ctp\tutorial2.cpp (where C:\FairCom\V* is your installation directory).
- Additional documentation may be found on the FairCom Web site at: www.faircom.com
2.3 Record/Row Locking

..\sdk\ctree.cpp\tutorials\ctpp_tutorial3.cpp

Now we will explore row/record locks using the c-treeACE C++ Database Framework.

The functionality for this tutorial focuses on inserting/adding rows/records, then updating a single row/record in the customer master table under locking control. The application will pause after a LOCK is placed on a row/record. Another instance of this application should then be launched, which will block, waiting on the lock held by the first instance. Pressing the <Enter> key will enable the first instance to proceed. This will result in removing the lock thereby allowing the second instance to continue execution. Launching two processes provides a visual demonstration of the effects of locking and a basis for experimentation on your own.

Like all other examples in the c-tree tutorial series, this tutorial simplifies the creation and use of a database into four simple steps: Initialize(), Define(), Manage(), and you’re Done() !

Tutorial #3: Locking

Here we demonstrate the enforcement of data integrity by introducing record/row "locking".

- Initialize() - Connects to the c-treeACE Database Engine.
- Define() - Defines and creates a "customer master" (custmast) table/file.
- Manage() - Adds a few rows/records; Reads the rows/records back from the database; displays the column/field content. Then demonstrates an update operation under locking control, and a scenario that shows a locking conflict.
- Done() - Disconnects from c-treeACE Database Engine.

Note our simple mainline:

```c
// main()
int main (COUNT argc, pTEXT argv[])
{
    #ifdef LOCK_SUPPORT
        Initialize();
        Define();
        Manage();
        Done();
    #else
        printf("This tutorial demonstrates basic Record Locking. Record Locking is not applicable to c-tree's SingleUser model. In order to run this program select a c-tree Multi-User or Client-Side type c-tree model");
        printf("Press <ENTER> key to exit . . .\n");
        getchar();
    #endif
```
return(0);
}

We suggest opening the source code with your own editor.

Continue now to review these four steps.
First we need to open a connection to a database by providing the c-treeACE Database Engine with a user name, password and the database name.

Below is the code for Initialize():

```c
// Initialize()
// Perform the minimum requirement of logging onto the c-tree Server

VOID Initialize(VOID)
{
    printf("INIT\n");
    try
    {
        // connect to server
        printf("\tLogon to server...\n");
        MySession.Logon("FAIRCOMS", ", ");
    }
    catch(CTException E)
    {
        Handle_Exception(E);
    }
}
```
Define

The define step is where specific data definitions are established by your application and/or process. This involves defining columns/fields and creating the tables/files with optional indices.

Below is the code for Define():

```c
//
// Define()
//
// Open the table, if it exists. Otherwise create and open the table
//
VOID Define(VOID)
{
    printf("DEFINE\n");
    Create_CustomerMaster_Table();
}

//
// Create_CustomerMaster_Table()
//
// Open table CustomerMaster, if it exists. Otherwise create it
// along with its indices and open it
//
VOID Create_CustomerMaster_Table(VOID)
{
    CTBOOL   do_create = NO;
    // define table CustomerMaster
    printf("\table CustomerMaster\n");
    try
    {
        MyTable.Open("custmast", CTOPEN_NORMAL);
    }
    catch (...)
    {
        // table does not exist
        do_create = YES;
    }
    if (do_create)
    {
        try
        {
            // define table fields
            CTFIELD field1 = MyTable.AddField("cm_custnumb", CT_FSTRING, 4);
            MyTable.AddField("cm_custzipc", CT_FSTRING, 9);
            MyTable.AddField("cm_custstat", CT_FSTRING, 2);
            MyTable.AddField("cm_custratg", CT_FSTRING, 1);
        }
    }
}
MyTable.AddField("cm_custname", CT_STRING, 47);
MyTable.AddField("cm_custaddr", CT_STRING, 47);
MyTable.AddField("cm_custcity", CT_STRING, 47);

// define index
CTIndex index1 = MyTable.AddIndex("cm_custnumb_idx", CTINDEX_FIXED, NO, NO);
index1.AddSegment(field1, CTSEG_SCHSEG);

// create table
printf("\tCreate table...\n");
MyTable.Create("custmast", CTCREATE_NORMAL);

// open table
printf("\tOpen table...\n");
MyTable.Open("custmast", CTOpen_NORMAL);

} catch (CTException E)
{
    Handle_Exception(E);
}
else
{
    Check_Table_Mode(MyTable);

    // confirm the index exists, if not then add the index
    //
    // this scenario arises out of the fact that this table was created in tutorial 1
    // without indices. The index is now created by the call to ctdbAlterTable

do_create = NO;
try
{
    MyTable.GetIndex("cm_custnumb_idx");
} catch (CTException E)
{
    do_create = YES;
}
if (do_create)
{
    try
    {
        CTFIELD field1 = MyTable.GetField("cm_custnumb");
        CTIndex index1 = MyTable.AddIndex("cm_custnumb_idx", CTINDEX_FIXED, NO, NO);
        index1.AddSegment(field1, CTSEG_SCHSEG);
        MyTable.Alter(CTDB.Alter_NORMAL);
    } catch (CTException E)
    {
        Handle_Exception(E);
    }

    if (do_create)
    {
        try
        {
            CTFIELD field1 = MyTable.GetField("cm_custnumb");
            CTIndex index1 = MyTable.AddIndex("cm_custnumb_idx", CTINDEX_FIXED, NO, NO);
            index1.AddSegment(field1, CTSEG_SCHSEG);
            MyTable.Alter(CTDB.Alter_NORMAL);
        } catch (CTException E)
        {
            Handle_Exception(E);
        }
    }
}

//
// Check_Table_Mode()
VOID Check_Table_Mode(CTTable& table)
{
  try
  {
    // get table create mode
    CTCREATE_MODE mode = table.GetCreateMode();

    // check if table is under transaction processing control
    if ((mode & CTCREATE_TRNLOG))
    {
      // change file mode to disable transaction processing
      mode ^= CTCREATE_TRNLOG;
      table.UpdateCreateMode(mode);
    }
  }
  catch (CTException E)
  {
    Handle_Exception(E);
  }
}
The manage step provides data management functionality for your application and/or process. 

Below is the code for `Manage()`:

```c
//
// Manage()
//
// This function performs record adds and updates using locking
//
VOID Manage(VOID)
{
    printf("MANAGE\n");
    // populate the table with data
    Add_CustomerMaster_Records();
    // display contents of table
    Display_Records();
    // update a record under locking control
    Update_CustomerMaster_Record();
    // display again after update and effects of lock
    Display_Records();
}
```

```c
//
// Add_CustomerMaster_Records()
//
// This function adds records to table CustomerMaster from an
// array of strings
//
VOID Add_CustomerMaster_Records(VOID)
{
    typedef struct {
        cpTEXT number, zipcode, state, rating, name, address, city;
    } DATA_RECORD;
    DATA_RECORD data[] = {
        "1000", "92867", "CA", "1", "Bryan Williams", "2999 Regency", "Orange",
        "1001", "61434", "CT", "1", "Michael Jordan", "13 Main", "Harford",
        "1003", "10034", "MO", "1", "Keyon Dooling", "19771 Park Avenue", "Columbia"
    };
    CTSIGNED nRecords = sizeof(data) / sizeof(DATA_RECORD);
    Delete_Records(MyRecord);
    Add_CustomerMaster_Records(nRecords);
    Display_Records();
}
printf("\tAdd records...\n");

try
{
    for (CTSIGNED i = 0; i < nRecords; i++)
    {
        MyRecord.Clear();

        // populate record buffer with data
        MyRecord.SetFieldAsString(0, data[i].number);
        MyRecord.SetFieldAsString(1, data[i].zipcode);
        MyRecord.SetFieldAsString(2, data[i].state);
        MyRecord.SetFieldAsString(3, data[i].rating);
        MyRecord.SetFieldAsString(4, data[i].name);
        MyRecord.SetFieldAsString(5, data[i].address);
        MyRecord.SetFieldAsString(6, data[i].city);

        // add record
        MyRecord.Write();
    }
}

catch(CTException E)
{
    Handle_Exception(E);
}


// Delete_Records()
// This function deletes all the records in the table
//
VOID Delete_Records(CTRecord& record)
{
    CTBOOL found;

    printf("\tDelete records...\n");

    try
    {
        // enable session-wide lock flag
        MySession.Lock(CTLOCK_WRITE_BLOCK);

        // read first record
        found = record.First();

        while (found) // while records are found
        {
            // delete record
            record.Delete();
            // read next record
            found = record.Next();
        }
    }
    catch(CTException E)
    {
        Handle_Exception(E);
    }
}
// Display_Records()
// This function displays the contents of a table. First() and Next() fetch the record. Then each field is parsed and displayed

VOID Display_Records(VOID)
{
    CTBOOL found;
    TEXT custnumb[4+1];
    TEXT custname[47+1];

    printf("\tDisplay records...");
    try
    {
        // read first record
        found = MyRecord.First();
        while (found)
        {
            strcpy(custnumb, MyRecord.GetFieldAsString(0).c_str());
            strcpy(custname, MyRecord.GetFieldAsString(4).c_str());

            // display data
            printf("\n\t%8s%10s\n", custnumb, custname);

            // read next record
            found = MyRecord.Next();
        }
    }
    catch(CTException E)
    {
        Handle_Exception(E);
    }
}

// Update_CustomerMaster_Records()
// Update one record under locking control to demonstrate the effects of locking

VOID Update_CustomerMaster_Record(VOID)
{
    printf("\tUpdate record...\n");
    try
    {
        // enable session-wide lock flag
        MySession.Lock(CTLOCK_WRITE_BLOCK);
        MyRecord.Clear();
        MyRecord.SetFieldAsString(0, "1003");
        // find record by customer number
if (MyRecord.Find(CTFIND_EQ))
{
    MyRecord.SetFieldAsString(4, "KEYON DOOLING");
    // rewrite record
    MyRecord.Write();

    printf("\tPress <ENTER> key to unlock\n");
    getchar();
}
}
catch(CTException E)
{
    Handle_Exception(E);
}

MySession.Unlock();
}
When an application and/or process has completed operations with the database, it must release resources by disconnecting from the database engine.

Below is the code for `Done()`:

```c
//
// Done()
//
// This function handles the housekeeping of closing, freeing,
// disconnecting and logging out of the database
//
VOID Done(VOID)
{
    printf("DONE\n");
    try
    {
        // close table
        printf("\tClose table...\n");
        MyTable.Close();
        // logout
        printf("\tLogout...\n");
        MySession.Logout();
    }
    catch(CTException E)
    {
        Handle_Exception(E);
    }
}
```
**Additional Resources**

We encourage you to explore the additional resources listed here:

- Complete source code for this tutorial can be found in `ctpp_tutorial3.cpp` in your installation directory, within the 'sdk\ctree.cpp\tutorials' directory for your platform. Example for the Windows platform: `C:\FairCom\V\win32\sdk\ctree.cpp\tutorials\ctpp_tutorial3.cpp` (where `C:\FairCom\V` is your installation directory).

- Additional documentation may be found on the FairCom Web site at: www.faircom.com
2.4 Transaction Processing

Now we will discuss transaction processing as it relates to the c-treeACE C++ Database Framework.

Transaction processing provides a safe method by which multiple database operations spread across separate tables/files are guaranteed to be atomic. By atomic, we mean that, within a transaction, either all of the operations succeed or none of the operations succeed. This "either all or none" atomicity ensures that the integrity of the data in related tables/files is secure.

Like all other examples in the c-tree tutorial series, this tutorial simplifies the creation and use of a database into four simple steps: Initialize(), Define(), Manage(), and You’re Done()!

Tutorial #4: Transaction Processing

Here we demonstrate transaction control.

- Initialize() - Connects to the c-treeACE Database Engine.
- Define() - Defines and creates our four tables/files.
- Manage() - Adds rows/records to multiple tables/files under transaction control.
- Done() - Disconnects from c-treeACE Database Engine.

Note our simple mainline:

```c
// main()

// The main() function implements the concept of "init, define, manage
// and you're done..."

int main (COUNT argc, pTEXT argv[])
{
    #ifdef TRANPROC
        Initialize();
        Define();
        Manage();
        Done();
    #else
        printf("\nThis tutorial demonstrates basic Transaction Processing. Transaction Processing is not");
        printf("\nactivated in your chosen c-tree's model. In order to run this program select a ");
        printf("\nc-tree model that supports Transaction processing, like the Client-Side c-tree model.");
    #endif

    printf("\nPress <ENTER> key to exit . . .\n");
    #ifndef ctPortWINCE
        getchar();
    #endif

    return(0);
}
```
We suggest opening the source code with your own editor.

Continue now to review these four steps.
Init

First we need to open a connection to a database by providing the c-treeACE Database Engine with a user name, password and the database name.

Below is the code for Initialize():

```c
// Initialize()
// Perform the minimum requirement of logging onto the c-tree Server

VOID Initialize(VOID)
{
    printf("INIT\n");

    // allocate the session object
    MySession = new CTSession(CTSESSION_CTREE);

    // allocate the table objects
    tableCustMast = new CTTable(MySession);
    tableCustOrdr = new CTTable(MySession);
    tableOrdrItem = new CTTable(MySession);
    tableItemMast = new CTTable(MySession);

    // allocate the record objects
    recordCustMast = new CTRecord(tableCustMast);
    recordCustOrdr = new CTRecord(tableCustOrdr);
    recordOrdrItem = new CTRecord(tableOrdrItem);
    recordItemMast = new CTRecord(tableItemMast);

    try
    {
        // connect to server
        printf("\tLogon to server...\n");
        MySession->Logon("FAIRCOMS", ",", ",");
    }
    catch (CTException E)
    {
        Handle_Exception(E);
    }
}
```
Define

The define step is where specific data definitions are established by your application and/or process. This involves defining columns/fields and creating the tables/files with optional indices.

Below is the code for Define():

```c
// Define()

VOID Define(VOID)
{
    printf("DEFINE\n");

    Create_CustomerMaster_Table();
    Create_CustomerOrders_Table();
    Create_OrderItems_Table();
    Create_ItemMaster_Table();
}

// Create_CustomerMaster_Table()

VOID Create_CustomerMaster_Table(VOID)
{
    CTBOOL   do_create = NO;
    // define table CustomerMaster
    printf("\table CustomerMaster\n");
    try
    {
        tableCustMast->Open("custmast", CTOPEN_NORMAL);
    }
    catch (CTException E)
    {
        // table does not exist
        do_create = YES;
    }

    if (do_create)
    {
        try
        {
            // define table fields
            CField field1 = tableCustMast->AddField("cm_custnumb", CT_FSTRING, 4);
        }
        catch (CTException E)
        {
            // handle exception
        }
    }
}
```
tableCustMast->AddField("cm_custzipc", CT_FSTRING, 9);
tableCustMast->AddField("cm_custstat", CT_FSTRING, 2);
tableCustMast->AddField("cm_custratg", CT_FSTRING, 1);
tableCustMast->AddField("cm_custname", CT_STRING, 47);
tableCustMast->AddField("cm_custaddr", CT_STRING, 47);
tableCustMast->AddField("cm_custcity", CT_STRING, 47);

// define index
CTIndex index1 = tableCustMast->AddIndex("cm_custnumb_idx", CTINDEX_FIXED, NO, NO);
index1.AddSegment(field1, CTSEG_SCHSEG);

// create table
printf("\tCreate table...\n");
tableCustMast->Create("custmast", CTCREATE_TRNLOG);

// open table
printf("\tOpen table...\n");
tableCustMast->Open("custmast", CTOPEN_NORMAL);
}
catch (CTException E)
{
    Handle_Exception(E);
}
else
{
    Check_Table_Mode(tableCustMast);

    // confirm the index exists, if not then add the index
    // this scenario arises out of the fact that this table was created in tutorial 1
    // without indices. The index is now created by the call to ctdbAlterTable

do_create = NO;
try
{
    tableCustMast->GetIndex("cm_custnumb_idx");
}
catch (CTException E)
{
    do_create = YES;
}
if (do_create)
{
    try
    {
        CTField field1 = tableCustMast->GetField("cm_custnumb");
        CTIndex index1 = tableCustMast->AddIndex("cm_custnumb_idx", CTINDEX_FIXED, NO, NO);
        index1.AddSegment(field1, CTSEG_SCHSEG);
        tableCustMast->Alter(CTDB_ALTER_NORMAL);
    }
catch (CTException E)
    {
        Handle_Exception(E);
    }
}
VOID Create_CustomerOrders_Table(VOID)
{
    CTBOOL   do_create = NO;

    // define table CustomerOrders
    printf("\ttable CustomerOrders\n");
    try
    {
        tableCustOrdr->Open("custordr", CTOPEN_NORMAL);
    }
    catch (CTException E)
    {
        // table does not exist
        do_create = YES;
    }

    if (do_create)
    {
        try
        {

            // define table fields
            tableCustOrdr->AddField("co_ordrdate", CT_DATE, 4);
            tableCustOrdr->AddField("co_promdate", CT_DATE, 4);
            CTFIELD field1 = tableCustOrdr->AddField("co_ordrnumb", CT_FSTRING, 6);
            CTFIELD field2 = tableCustOrdr->AddField("co_custnumb", CT_FSTRING, 4);

            // define indices
            CTINDEX index1 = tableCustOrdr->AddIndex("co_ordrnumb_idx", CTINDEX_LEADING, NO, NO);
            index1.AddSegment(field1, CTSEG_SCHSEG);
            CTINDEX index2 = tableCustOrdr->AddIndex("co_custnumb_idx", CTINDEX_LEADING, YES, NO);
            index2.AddSegment(field2, CTSEG_SCHSEG);

            // create table
            printf("\tCreate table...\n");
            tableCustOrdr->Create("custordr", CTCREATE_TRNLOG);

            // open table
            printf("\tOpen table...\n");
            tableCustOrdr->Open("custordr", CTOPEN_NORMAL);
        }
        catch (CTException E)
        {
            Handle_Exception(E);
        }
    }
    else
    {
        Check_Table_Mode(tableCustOrdr);
    }

    // Create_OrderItems_Table()
    // Open table OrderItems, if it exists. Otherwise create it
VOID Create_OrderItems_Table(VOID)
{
    CTBOOL   do_create = NO;

    // define table OrderItems
    printf("\table OrderItems\n");
    try
    {
        tableOrdrItem->Open("ordritem", CTOPEN_NORMAL);
    }
    catch (CTException E)
    {
        // table does not exist
        do_create = YES;
    }

    if (do_create)
    {
        try
        {
            // define table fields
            CTField field1 = tableOrdrItem->AddField("oi_sequnumb", CT_INT2, 2);
            field1 = tableOrdrItem->AddField("oi_quantity", CT_INT2, 2);
            CTField field2 = tableOrdrItem->AddField("oi_ordrnumb", CT_FSTRING, 6);
            field2 = tableOrdrItem->AddField("oi_itemnumb", CT_FSTRING, 5);

            // define indices
            CTDINDEX idx1 = tableOrdrItem->AddIndex("oi_ordrnumb_idx", CTINDEX_LEADING, NO, NO);
            index1.AddSegment(field2, CTSEG_SCHSEG);
            index1.AddSegment(field1, CTSEG_SCHSEG);
            CTDINDEX idx2 = tableOrdrItem->AddIndex("oi_itemnumb_idx", CTINDEX_LEADING, YES, NO);
            index2.AddSegment(field3, CTSEG_SCHSEG);

            // create table
            printf("\tCreate table...
");
            tableOrdrItem->Create("ordritem", CTCREATE_TRNLOG);

            // open table
            printf("\tOpen table...
");
            tableOrdrItem->Open("ordritem", CTOPEN_NORMAL);
        }
        catch (CTException E)
        {
            Handle_Exception(E);
        }
    }
    else
    {
        Check_Table_Mode(tableOrdrItem);
    }
}

// Create_ItemMaster_Table()
//
// Open table ItemMaster, if it exists. Otherwise create it
// along with its indices and open it
//
VOID Create_ItemMaster_Table(VOID)
{
    CTBOOL do_create = NO;

    // define table ItemMaster
    printf("\ttable ItemMaster\n");
    try
    {
        tableItemMast->Open("itemmast", CTOPEN_NORMAL);
    }
    catch (CTException E)
    {
        // table does not exist
        do_create = YES;
    }

    if (do_create)
    {
        try
        {
            // define table fields
            tableItemMast->AddField("im_itemwght", CT_INT4, 4);
            tableItemMast->AddField("im_itempric", CT_MONEY, 4);
            CTField field1 = tableItemMast->AddField("im_itemnumb", CT_FSTRING, 5);
            tableItemMast->AddField("im_itemdesc", CT_STRING, 47);

            // define indices
            CTIndex index1 = tableItemMast->AddIndex("im_itemnumb_idx", CTINDEX_FIXED, NO, NO);
            index1.AddSegment(field1, CTSEG_SCHSEG);

            // create table
            printf("\tCreate table...");
            tableItemMast->Create("itemmast", CTCREATE_TRNLOG);

            // open table
            printf("\tOpen table...");
            tableItemMast->Open("itemmast", CTOPEN_NORMAL);
        }
        catch (CTException E)
        {
            Handle_Exception(E);
        }
    }
    else
    {
        Check_Table_Mode(tableItemMast);
    }
}

// Check_Table_Mode()
//
// Check if existing table has transaction processing flag enabled.
// If a table is not ready for transaction, modify the table mode to
// enable transaction processing
//
VOID Check_Table_Mode(CTTable* table)
{
    try
    {
        // get table create mode
        //
CTCREATE_MODE mode = table->GetCreateMode();

// check if table is not under transaction processing control
if (!(mode & CTCREATE_TRNLOG))
{
    // change file mode to enable transaction processing
    mode |= CTCREATE_TRNLOG;
    table->UpdateCreateMode(mode);
}

} catch (CTException E)
{
    Handle_Exception(E);
}
The manage step provides data management functionality for your application and/or process.

Below is the code for `Manage()`:

```c
//
// Manage()
//
// Populates table and perform a simple query
//
VOID Manage(VOID)
{
    printf("MANAGE\n");

    // populate the tables with data
    Add_CustomerMaster_Records();
    Add_ItemMaster_Records();
    Add_Transactions();

    // display the orders and their items
    Display_CustomerOrders();
    Display_OrderItems();
}

//
// Add_CustomerMaster_Records()
//
// This function adds records to table CustomerMaster from an
// array of strings
//
VOID Add_CustomerMaster_Records(VOID)
{
    typedef struct {
        cpTEXT number, zipcode, state, rating, name, address, city;
    } DATA_RECORD;

    DATA_RECORD data[] = {
        "1000", "92867", "CA", "1", "Bryan Williams", "2999 Regency", "Orange",
        "1001", "61434", "CT", "1", "Michael Jordan", "13 Main", "Harford",
        "1003", "10034", "MO", "1", "Keyon Dooling", "19771 Park Avenue", "Columbia"
    };

    CTSIGNED nRecords = sizeof(data) / sizeof(DATA_RECORD);
    Delete_Records(recordCustMast);

    printf("\tAdd records in table CustomerMaster...\n");
```
try
{
    // start a transaction
    recordCustMast->Begin();

    for (CTSIGNED i = 0; i < nRecords; i++)
    {
        recordCustMast->Clear();

        // populate record buffer with data
        recordCustMast->SetFieldAsString(0, data[i].number);
        recordCustMast->SetFieldAsString(1, data[i].zipcode);
        recordCustMast->SetFieldAsString(2, data[i].state);
        recordCustMast->SetFieldAsString(3, data[i].rating);
        recordCustMast->SetFieldAsString(4, data[i].name);
        recordCustMast->SetFieldAsString(5, data[i].address);
        recordCustMast->SetFieldAsString(6, data[i].city);

        // add record
        recordCustMast->Write();
    }

    // commit transaction
    recordCustMast->Commit();
}
catch(CTException E)
{
    Handle_Exception(E);
}

// Add_ItemMaster_Records()
// This function adds records to table ItemMaster from an
// array of strings
//
VOID Add_ItemMaster_Records(VOID)
{
    typedef struct {
        CTSIGNED weight;
        CTMONEY price;
        cpTEXT itemnum, description;
    } DATA_RECORD;

    DATA_RECORD data[] = {
        {10, 1995, "1", "Hammer"},
        {3, 999, "2", "Wrench"},
        {4, 1659, "3", "Saw"},
        {1, 398, "4", "Pliers"}
    };  
    CTSIGNED nRecords = sizeof(data) / sizeof(DATA_RECORD);
    Delete_Records(recordItemMast);
    printf("\tAdd records in table ItemMaster...\n");
    try
    {
        // //
// start a transaction
recordItemMast->Begin();

for (CTSigned i = 0; i < nRecords; i++)
{
    recordItemMast->Clear();

    // populate record buffer with data
    recordItemMast->SetFieldAsSigned(0, data[i].weight);
    recordItemMast->SetFieldAsMoney(1, data[i].price);
    recordItemMast->SetFieldAsString(2, data[i].itemnum);
    recordItemMast->SetFieldAsString(3, data[i].description);

    // add record
    recordItemMast->Write();
}

    // commit transaction
    recordItemMast->Commit();
}
catch(CTException E)
{
    Handle_Exception(E);
}

// Delete_Records()
// This function deletes all the records in the table
//
VOID Delete_Records(CTRecord* record)
{
    CTBOOL    found;
    printf("\tDelete records...\n");
    try
    {
        // write lock required for transaction updates
        record->Lock(CTLOCK_WRITE);

        // start a transaction
        record->Begin();

        // read first record
        found = record->First();

        while (found)  // while records are found
        {
            // delete record
            record->Delete();
            // read next record
            found = record->Next();
        }

        // commit transaction
        record->Commit();
    }
    catch(CTException E)
    {
        Handle_Exception(E);
    }
}
// free locks
    record->Unlock();
}
catch(CTException E)
{
    Handle_Exception(E);
}
}

// Add_Transactions()
// Add an Order and associated Items "as a transaction" to their
// respective tables. A transaction is committed or aborted if the
// customer number on the order is confirmed valid. Likewise each
// item in the order is verified to be a valid item. SavePoints are
// established as an order is processed, allowing a transaction to
// rollback to the previously verified item

VOID Add_Transactions(VOID)
{
    typedef struct {
        cpTEXT orderdate, promdate, ordernum, custnum;
    } ORDER_DATA;

    typedef struct {
        cpTEXT ordernum;
        CTSIGNED seqnumber;
        CTSIGNED quantity;
        cpTEXT itemnum;
    } ITEM_DATA;

    ORDER_DATA orders[] = {
        {"09/01/2002", "09/05/2002", "1", "1001"},
        {"09/02/2002", "09/06/2002", "2", "9999"}, // bad customer number
        {"09/22/2002", "09/26/2002", "3", "1003"}
    };

    ITEM_DATA items[] = {
        {"1", 1, 2, "1"},
        {"1", 2, 1, "2"},
        {"2", 2, 3, "4"},
        {"3", 1, 2, "3"},
        {"3", 2, 2, "99"} // bad item number
    };

    CTSIGNED nOrders = sizeof(orders) / sizeof(ORDER_DATA);
    CTSIGNED nItems = sizeof(items) / sizeof(ITEM_DATA);
    NINT  savepoint;
    CTDate orderdate;
    CTDate promdate;
    CTSIGNED j = 0;

    Delete_Records(recordCustOrdr);
    Delete_Records(recordOrdrItem);

    printf("\tAdd transaction records... \n");

    // process orders
for (CTSIGNED i = 0; i < nOrders; i++)
{
    // start a transaction
    MySession->Begin();

    try
    {
        recordCustOrdr->Clear();

        // populate record buffer with order data
        orderdate.StringToDate(orders[i].orderdate, CDATE_MDCY);
        promdate.StringToDate(orders[i].promdate, CDATE_MDCY);
        recordCustOrdr->SetFieldAsDate(0, orderdate);
        recordCustOrdr->SetFieldAsDate(1, promdate);
        recordCustOrdr->SetFieldAsString(2, orders[i].ordernum);
        recordCustOrdr->SetFieldAsString(3, orders[i].custnum);

        // add order record
        recordCustOrdr->Write();
    }
    catch (CTException E)
    {
        Handle_Exception(E);
    }

    // set transaction savepoint
    savepoint = recordCustOrdr->SetSavePoint();

    // process order items
    while (!(strcmp(items[j].ordernum, orders[i].ordernum)))
    {
        try
        {
            recordOrdrItem->Clear();

            // populate record buffer with order item data
            recordOrdrItem->SetFieldAsSigned(0, items[j].seqnumber);
            recordOrdrItem->SetFieldAsSigned(1, items[j].quantity);
            recordOrdrItem->SetFieldAsString(2, items[j].ordernum);
            recordOrdrItem->SetFieldAsString(3, items[j].itemnum);

            // add order item record
            recordOrdrItem->Write();

            // check that item exists in ItemMaster table
            recordItemMast->Clear();
            recordItemMast->SetFieldAsString(2, items[j].itemnum);
            if (!recordItemMast->Find(CTFIND_EQ))
                // if not found, restore back to previous savepoint
                recordItemMast->RestoreSavePoint(savepoint);
            else
                // set transaction savepoint
                savepoint = recordItemMast->SetSavePoint();
        }
        catch (CTException E)
        {
            Handle_Exception(E);
        }
    }

    // bump to next item
    j++;
}
// exit the while loop on last item
if (j >= nItems)
    break;
}

// check that customer exists in CustomerMaster table
recordCustMast->Clear();
recordCustMast->SetFieldAsString(0, orders[i].custnum);

// commit or abort the transaction
if (!recordCustMast->Find(CTFIND_EQ))
    recordCustMast->Abort();
else
    recordCustMast->Commit();
}
}

// Display_CustomerOrders()
//
// This function displays the contents of a table. ctdbFirstRecord() and
cmdbNextRecord() fetch the record. Then each field is parsed and displayed

VOID Display_CustomerOrders(VOID)
{
    CTString custnum;
    CTString ordrnum;

    printf("\n\tCustomerOrder table...
\n");

    try
    {
        // read first record
        if (recordCustOrdr->First())
        {
            do
            {
                ordrnum = recordCustOrdr->GetFieldAsString(2);
                custnum = recordCustOrdr->GetFieldAsString(3);

                // display data
                printf("\t %s %s\n", ordrnum.c_str(), custnum.c_str());
            }
            // read next record until end of file
            while (recordCustOrdr->Next());
        }
        catch (CTException E)
        {
            Handle_Exception(E);
        }
    }

    // Display_OrderItems()
    //
    // This function displays the contents of a table. ctdbFirstRecord() and
VOID Display_OrderItems(VOID)
{
    CTString itemnumb;
    CTString ordrnumb;

    printf("\n\tOrderItems Table...
\n");

    try
    {
        // read first record
        if (recordOrdrItem->First())
        {
            do
            {
                // get field data from record buffer
                ordrnumb = recordOrdrItem->GetFieldAsString(2);
                itemnumb = recordOrdrItem->GetFieldAsString(3);

                // display data
                printf("\t %s   %s\n", ordrnumb.c_str(), itemnumb.c_str());
            }
            // read next record until end of file
            while (recordOrdrItem->Next());
        }
    }
    catch (CTException E)
    {
        Handle_Exception(E);
    }
}
When an application and/or process has completed operations with the database, it must release resources by disconnecting from the database engine.

Below is the code for **Done()**:

```c
//
// Done()
//
// This function handles the housekeeping of closing, freeing,
// disconnecting and logging out of the database
//

VOID Done(VOID)
{
    printf("DONE\n");

    try
    {
        // close tables
        printf("\tClose tables...\n");
        tableCustOrdr->Close();
        tableOrdrItem->Close();
        tableItemMast->Close();
        tableCustMast->Close();

        // logout from session
        printf("\tLogout...
");
        MySession->Logout();
    }
    catch(CTException E)
    {
        Handle_Exception(E);
    }

    // release record objects
    delete recordCustMast;
    delete recordItemMast;
    delete recordOrdrItem;
    delete recordCustOrdr;

    // release table objects
    delete tableCustMast;
    delete tableItemMast;
    delete tableOrdrItem;
    delete tableCustOrdr;

    // release session object
    delete MySession;
}
```
Additional Resources

We encourage you to explore the additional resources listed here:

- Complete source code for this tutorial can be found in ctpp_tutorial4.cpp in your installation directory, within the 'sdk\ctree.cpp\tutorials' directory for your platform.
  Example for the Windows platform:
  C:\FairCom\V*\win32\sdk\ctree.cpp\tutorials\ctpp_tutorial4.cpp (where C:\FairCom\V* is your installation directory).
- Additional documentation may be found on the FairCom Web site at: www.faircom.com
c-treeDB, short for c-tree DataBase, represents a higher-level, easier to use API on top of the two popular FairCom APIs: ISAM and Low-level. c-treeDB is intended as the new standard for c-tree Plus programming. FairCom tried to make the developer’s life easier without removing the flexibility and performance of the original APIs.

The c-treeDB general architecture is presented in the figure below, organized into seven different levels: session, database, table, field, index, segment, and record. These levels or layers will be used to present a group of common functionality.

It is important to note that c-tree data and index files can be manipulated directly with or without session or database dictionary support. Please refer to "Working with Sessions without Dictionary Support" (page 74) and "Creating a table object without database support" (page 86) for more information.

**c-treeDB Relationships**

A **Session** represents a connection between a client and a c-tree Server; no work can be performed before a session becomes active. The session object indicates the c-treeDB session, the server name and location, the directory where the databases are located, the user name and password.

A **Database** can be considered as a collection of tables, and each database has its own database dictionary that stores information about each table that belongs to that database: the table name, password and path, the active (open) tables, and the number of tables linked to the database. The database object indicates a database in the session and each session can have multiple databases.

A **Table** is essentially a c-tree Plus data file and optional index files. There can be, and typically are, more than one table in a database, and a given table may belong to multiple databases. A Table may have zero or more records.

A **Field** is the basic element of a table, and a collection of fields form a data record.
Often a table will have zero or more **Indices**, which enhances the retrieval of records from that table.

Indices typically have one or more **Segments** that describe the index key structure. The index object indicates an index associated with a particular table, while the segment links the index with the fields.

A **Record** is essentially an entry row in a table. A record object indicates a record instance on a particular table. A table may have one or more record objects associated with it. Each record handle may be an independent cursor into the table, or several record objects may share the same cursor into the table.

### 3.1 Common functionality

c-treeDB classes **CTSession**, **CTDatabase**, **CTTable**, **CTRecord**, **CTField**, **CTIndex** and **CTSegment** all inherit from a common base **CTBase** class.

All functionality and methods described in this chapter are availed when working with sessions, databases, tables, records, fields, indices and index segments.

**Error Handling**

Most c-treeDB API functions return an error status to indicate if a particular function operation succeeded or not. Most c-treeDB API methods will also keep the last error status, if the function operation failed. The last error status can be manipulated with the following methods:

- **GetError()** retrieves the last error status. If a function succeeds, the success status is not kept by the c-treeDB API.
- **SetError()** sets the last error status, overwriting any previous error status kept by the c-treeDB API.
- **ClearError()** clears the last error status.

```c
// clear the error if error is INOT_ERR
// Recobj is a CTRecord object
if (Recobj.GetError() == INOT_ERR)
{
    Recobj.ClearError();
}
```

**Transaction Processing**

The c-treeDB API implementation of transaction processing functions follows closely the c-tree Plus ISAM API definition. The basic difference between the c-treeDB and ISAM transaction processing API is the separation of locking mechanisms from transaction processing.

While the ISAM API also allows for locking modes to be specified when beginning a transaction, the c-treeDB API requires that locking functions be implicitly used after a transaction begins. The code fragment below shows an example using a c-tree Plus ISAM call to start a transaction with locking:

```c
/* start transaction enabling write locks */
TRANBEG(TRNLOG | ENABLE | LK_BLOCK);
```
... perform some data operations ...

/* commit the transaction and release locks */
TRANEND(FREE);

The c-tree Plus code fragment above starts a transaction by invoking function TRANBEG. The mode ENABLE indicates that every c-tree Plus read record operation will lock the record for writing and the mode LK_BLOCK indicates that the thread will block until the lock is acquired.

When using the c-treeDB API, users must be aware that the transaction processing API will not start the record locking mechanism. The code fragment below shows the equivalent example using the c-treeDB API:

// Recobj is a CTRecord object

// start a transaction
Recobj.Begin();

// enable write locks
Recobj.Lock(CTLOCK_WRITE_BLOCK);
... perform some data operations ...

// release locks
Recobj.Unlock();

// commit the transaction
Recobj.Commit();

The Begin() method starts a new transaction, while Commit() terminates a transaction by committing any changes. Abort() terminates a transaction and aborts any changes done during the transaction.

SetSavePoint() sets a save point in the current transaction. Once a save point is set within the current transaction, RestoreSavePoint() will reverse only changes done in between the set save point and restore the save point, without terminating the transaction.

Please refer to "Data Integrity" in the c-tree Plus Programmer’s Reference Guide for a detailed description of transaction processing and locking.

## Session Wide Locking

Session wide locking is based on the principle that the Lock() method sets the current lock mode. When locks are activated, every record read from all active tables of all active databases associated with the session are automatically locked with the current lock mode.

// start locking
// ASession is a CTSession object
try
{
    ASession.Lock(CTLOCK_WRITE_BLOCK);
}
catch (CTException &err)
{
    printf("Session lock failed with error %d\n", err.GetErrorCode());
}

Unlock() releases all locks acquired since the last Lock() call and clears the current lock mode. IsLockActive() indicates if a session wide lock mode is set. GetLockMode() retrieves the current
session wide lock mode. If no session wide locks are active, `GetLockMode()` returns `CTLOCK_FREE`.

```cpp
// unlock if locks are active
// ARecord is a CTRecord object
try {
    if (ARecord.GetLockMode() != CTLOCK_FREE)
    {
        ARecord.Unlock();
    }
} 
catch (CTException &err)
{
    printf("Unlock failed with code %d\n", err.GetErrorCode());
}
```

Please refer "Date Integrity" in `c-tree Plus Programmer's Reference Guide` for a detailed description of transaction processing and locking.

**Default Date, Time and Float formats**

The `c-treeDB` record manager performs automatic data type conversions when the user reads from, or writes to, a field using a data type that is different from the field data type. For most data types, the conversion is straightforward except when converting dates and times to and from strings, as there are many different conventions for displaying dates and times.

By default `c-treeDB` converts date to string, and from string to date, using the standard USA convention of MM/DD/CCYY, where MM represents a two digit month with values from 01 to 12, DD represents a two digit day of the month with values from 01 to 31, depending on the number of days in the month, CC represents a two digit century and YY represents a two digit year. A date separator may be the ‘/’, ‘-’ and ‘.’ characters.

The `c-treeDB` API converts time to string, and string to time, using the standard USA convention of HH:MM AM where HH represents the hour with values from 1 to 12, MM represents the minutes with values from 1 to 59 and AM represents AM or PM values.

**Date Formats**

`SetDefDateFormat()` sets a new default date format. `GetDefDateFormat()` retrieves the current default date format. The following date formats are supported:

- **CTDATE_MDCY** Date format is MM/DD/CCYY where MM represents a two-digit month, DD represents a two-digit day of the month, CC represents a two-digit century, and YY represents a two-digit year. The date separator may be one of the following characters: ‘/’, ‘-’ or ‘.’. This is the default date format. Example: 12/01/2002.

- **CTDATE_MDY** Date format is MM/DD/YY where MM represents a two-digit month, DD represents a two-digit day of the month, and YY represents a two-digit year. The date separator may be one of the following characters: ‘/’, ‘-’ or ‘.’. Example: 12/01/2002.

- **CTDATE_DMCY** Date format is DD/MM/CCYY where DD represents a two-digit day, MM represents a two-digit month, CC represents a two-digit century, and YY represents a two-digit year. The date separator may be one of the following characters: ‘/’, ‘-’ or ‘.’. Example: 01/12/2002.
• **CTDATE_DMY**  Date format is DD/MM/YY where DD represents a two-digit day, MM represents a two-digit month, and YY represents a two-digit year. The date separator may be one of the following characters: ‘/’, ‘-’ or ‘.’. Example: 01/12/02.

• **CTDATE_CYMD**  Date format is CCYYMMDD where CC is a two-digit century, YY is a two-digit date, MM is a two-digit month, and DD is a two-digit day of the month. This date format has no separators. Example: 20021201.

• **CTDATE_YMD**  The date format is YYMMDD where YY represents a two-digit year, MM represents a two-digit month, and DD represents a two-digit day of the month. This date format has no separators. Example: 021201

### Time Formats

**SetDefTimeFormat()** sets a new default time format. **GetDefTimeFormat()** retrieves the current default time format. The following time formats are supported:

• **CTTIME_HMSP**  Time format is HH:MM:SS AP where HH represents an hour value between 1 and 12, MM represents a two-digit minute value between 00 and 59, SS represents a two-digit second value between 00 and 59, and AP is either AM or PM. The time separator may be ‘:’ or ‘.’. Example: 1:35:45 AM.

• **CTTIME_HMP**  Time format is HH:MM AP where HH represents an hour value between 1 and 12, MM represents a two-digit minute value between 00 and 59, and AP is either AM or PM. The time separator may be ‘:’ or ‘.’. Example: 1:35 AM.

• **CTIME_HMS**  Time format is HH:MM:SS where HH represents an hour value between 0 and 23, MM represents a two-digit minute value between 00 and 59, and SS represents a two-digit second value between 00 and 59. The time separator may be ‘:’ or ‘.’. Example: 1:35:45.

• **CTIME_HM**  Time format is HH:MM where HH represent an hour value between 0 and 23, MM represents a two-digit minute value between 00 and 59. The time separator may be ‘:’ or ‘.’. Example: 1:35.

• **CTTIME_MIL**  Time format is HHMM (military format). HH represents a two-digit hour value between 00 and 23 and MM represents a two-digit minute value between 00 and 59. This time format has no separator. Example: 0135.

### Float Formats

When converting floating point type fields, such as **CT_SFLOAT**, **CT_DFLOAT**, and **CT_EFLOAT**, to and from strings, c-treeDB uses the float conversion format used by the C standard library functions **printf()** and **scanf()**. By default the float conversion format is set to “%f”. Use **SetDefFloatFormat()** set a new default float conversion format. Use **GetDefFloatFormat()** to retrieve the current default float conversion format.

### User Defined Tags

Every handle allocated by the c-treeDB API has a space called user tag value reserved for holding an arbitrary user value. The user tag has no predefined meaning and is provided for the convenience of developers. It can be used for storing an additional void pointer or it can be typecast to any 32-bit value (or 64-bit value on 64-bit platforms).
Use `GetUserTag()` to retrieve the current user tag value associated with a handle. Use `SetUserTag()` to set the current user tag value associated with a handle.

When a c-treeDB object is destroyed, the user tag value is not automatically released. The user is responsible for releasing any dynamic memory controlled by pointers stored in the handle user tag space.

### 3.2 Working with Sessions

The c-treeDB interface requires a session object to perform any data structure initialization or manipulation. The following steps must be executed within a session before any database or table operations are attempted:

1. Construct a `CTSession` object
2. Logon to a c-tree session by calling the `CTSession::Logon()` method.

Then, after this initialization, all database and table operations should be performed. After these, in order to finalize the database operations, the following steps should be performed:

1. Logout from a c-tree session by calling the `CTSession::Logout()` method.
2. Destroy the `CTSession` object

#### Creating a Session object

A valid session object is required to perform any session operation. The default parameter of the `CTSession` constructor is `CTSESSION_CTDB`.

```cpp
// create a default CTSession object
CTSession ASession;
try
{
    // logon to session using default
    // server name, user name and password.
    CTSession.Logon();
} catch (CTException &err) {
    printf("Session logon failed with error %d\n", err.GetErrorCode);
}
```

If you create a dynamic `CTSession` object with the new operator, you are required to destroy the object with the delete operator.

```cpp
// create a dynamic CTSession object
CTSession* pSession = new CTSession;
if (!pSession) {
    printf("CTSession allocation failed\n");
} ... other operations ..
// destroy the CTSession object
delete pSession;
```
When a session object is created, we can specify the session type. There are three different session types:

<table>
<thead>
<tr>
<th>CTSESSION_CTREE</th>
<th>Allocate a new session for logon only. No session or database dictionary files will be used. No database functions can be used with this session mode. You must allocate table handles using the session handle.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTSESSION_CTB</td>
<td>Allocate a new session making full usage of c-treeDB session and database dictionaries. With this session mode, a Session dictionary file must exist to perform a session logon and you need to connect to a database before attempting to perform operation on tables.</td>
</tr>
<tr>
<td>CTSESSION_SQL</td>
<td>Allocate a new session for c-treeACE SQL processing. This mode allows changes made at the c-treeDB level to be reflected at the c-treeACE SQL level without requiring an additional table import step; your tables are immediately available with the c-treeACE SQL interface. While this session type is available, limitations still exist. Support for alter table operations are not supported; c-treeDB AlterTable() activities are NOT reflected in the c-treeACE SQL system tables.</td>
</tr>
</tbody>
</table>

// Create a session object
CTSession ASession(CTSESSION_CTB);

**Creating a new session dictionary**

When operating with sessions of type CTSESSION_CTB or CTSESSION_SQL, a session dictionary file named ctdbdict.fsd is required before a session logon is performed. The session dictionary file ctdbdict.fsd is located by default either in the server directory (client/server application) or in the application directory (stand-alone application).

There could be situations where the c-treeACE SQL session dictionary file does not exist because it was deleted or this is the very first time the system is being executed. In this case it is necessary to create a session dictionary file before attempting to logon to a session. It is important to note that only one session dictionary file is necessary for the normal operation of c-treeACE SQL. Once created, the session dictionary file can be used for all database and table operations.

The following code fragment shows an example on how to create a new session dictionary file:

CTSession ASession;
try
{
  ASession.Create("FAIRCOMS", "ADMIN", "ADMIN");
}
catch (CTException &err)
{
  printf("Create session failed with error %d\n", Err.GetErrorCode());
}

In the server name, the full network address may be used (FAIRCOMS@10.0.0.1 or FAIRCOMS@my_machine.com for instance). For non-server applications, the parameter is ignored and may be set to NULL.

A valid user name is required to access the c-tree Server. When the server is first installed, only the default user is permitted server access. This user name, ADMIN, is intended for the database administrator. ADMIN is also the default user name for c-treeDB sessions. For non-server applications, the user name may be NULL. For server applications, see the c-tree Server Administrator’s Guide on how to create users and groups.
A valid user password is also required to access the c-tree Server. When the server is first installed, the default user ADMIN is associated with the default password, also ADMIN. For non-server applications, the user password may be NULL.

**Session logon and logout**

In order to perform any database operations, it is necessary to logon to a c-tree session. A session is terminated with a session logout.

To log on to a session, a *CTSession* object must be instantiated and then *CTSession::Logon()* method should be called to perform the session logon.

**Example**

```cpp
// Logon to a session
CTSession ASession;
try
{
    ASession.Logon("FAIRCOMS", "ADMIN", "ADMIN");
} catch (CTException &err)
{
    printf("Session logon failed with error %d\n", err.GetErrorCode());
}
```

**Note:** A useful sequence in code is to try to log on to the session, and if it fails with error FNOP_ERR (12), create the session dictionary and then log on again.

**Example:**

```cpp
CTSession ASession;
CTBOOL createit = NO;
// try to logon to a session
try
{
    ASession.Logon("FAIRCOMS", "ADMIN", "ADMIN");
} catch (CTException &err)
{
    if (err.GetErrorCode() == FNOP_ERR)
    {
        createit = YES;
    } else
    {
        throw;
    }
    if (createit)
    {
        ASession.Create("FAIRCOMS", "ADMIN", "ADMIN");
        try
        {
            ASession.Logon("FAIRCOMS", "ADMIN", "ADMIN");
        } catch (CTException &err)
        {
        }
```
When operations with the session are no longer needed, it is necessary to logout from the session by invoking method `CTSession::Logout()`.

**Example**

```cpp
CTSession ASession;
// logon to a session
try {
    ASession.Logon("FAIRCOMS", "ADMIN", "ADMIN");
    // ... perform some other operations ..
    ASession.Logout();
} catch (CTException &err) {
    printf("Logon or Logout failed with error %d\n", err.GetErrorCode());
}
```

**Session properties**

The default session properties are suitable for most c-treeDB applications. Advanced developers may need to tune c-treeDB to meet application requirements. Use `CTSession::SetSessionParams()` to set the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Explanation</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BUFS</code></td>
<td>Index file buffers</td>
<td>10</td>
</tr>
<tr>
<td><code>FILS</code></td>
<td>File structure blocks</td>
<td>32</td>
</tr>
<tr>
<td><code>SECT</code></td>
<td>Node sectors</td>
<td>32</td>
</tr>
<tr>
<td><code>DBUFS</code></td>
<td>Data file buffers</td>
<td>10</td>
</tr>
<tr>
<td><code>USERPROF</code></td>
<td>User profile mask</td>
<td>513</td>
</tr>
</tbody>
</table>

The default `USERPROF` value of 513 tells single-user, transaction-control applications, to remove the auxiliary log files `S*.FCS` and `L*.FCS` upon successful termination of the application, and also removes the automatic key transformation.

The table below present all possible values for the `USERPROF` parameter.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USERPRF_NTKEY</code></td>
<td>1</td>
<td>Do not perform auto tfrmkey</td>
</tr>
<tr>
<td><code>USERPRF_SAVENV</code></td>
<td>2</td>
<td>Savenv mode for transactions</td>
</tr>
<tr>
<td>Keyword</td>
<td>Value</td>
<td>Explanation</td>
</tr>
<tr>
<td>----------------</td>
<td>-------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>USERPRF_SQL</td>
<td>4</td>
<td>Enable SQL support</td>
</tr>
<tr>
<td>USERPRF_SERIAL</td>
<td>8</td>
<td>Enable strict serialization</td>
</tr>
<tr>
<td>USERPRF_MEMABS</td>
<td>10</td>
<td>Internal memory management</td>
</tr>
<tr>
<td>USERPRF_NDATA</td>
<td>32</td>
<td>Do not perform auto data - UNIFRMAT conversion</td>
</tr>
<tr>
<td>USERPRF_LOCLIB</td>
<td>64</td>
<td>Use a local library: not server</td>
</tr>
<tr>
<td>USERPRF_PTHTMP</td>
<td>128</td>
<td>Add tmpname to input path, otherwise use system tmpname</td>
</tr>
<tr>
<td>USERPRF_CODCNV</td>
<td>256</td>
<td>Auto language conversion</td>
</tr>
<tr>
<td>USERPRF_CLRCHK</td>
<td>512</td>
<td>Clear transaction logs</td>
</tr>
<tr>
<td>USERPRF_CUSTOM</td>
<td>1024</td>
<td>Custom server application</td>
</tr>
<tr>
<td>USERPRF_ENCRYPT</td>
<td>2048</td>
<td>Enable encryption</td>
</tr>
<tr>
<td>USERPRF_ADMSPCL</td>
<td>4096</td>
<td>Full admin logon</td>
</tr>
</tbody>
</table>

**Example**

```c++
// set a different user profile before logging on to a session
CTSession ASession;
try {
    // set the new profile
    ASession.SetSessionParam(USERPROF, (USERPRF_NTKEY | USERPRF_CLRCHK));

    // logon to a session
    ASession.Logon("FAIRCOMS", "ADMIN", "ADMIN");
} catch (CTException &err) {
    printf("Session logon failed with error %d\n", err.GetErrorCode());
}
```

**Server, user name and password properties**

Once a session is active, the user may retrieve the server name for the session with `CTSession::GetServerName()`. Retrieve the user name and the user password by calling the `CTSession::GetUserLogonName()` and `CTSession::GetUserPassword()` methods respectively. Please refer to the example in "Active property" (page 69).

**Active property**

A session is active if the user has logged on to a valid session dictionary using `CTSession::Logon()`. To render a session inactive, log off using `CTSession::Logout()`. To verify the status of the session, use `CTSession::IsActive()`.

**Example**
// if session is active, retrieve the server name, user logon name and user password
if (ASession.IsActive())
{
    printf("Server name: %s\n", ASession.GetServerName());
    printf("User name  : %s\n", ASession.GetUserLogonName());
    printf("Password   : %s\n", ASession.GetPassword());
}

Path property
The default path for client/server applications is the server directory, while the default path for non-server applications is the application directory. If, for any reason, there is the need to modify the location of the session dictionary, the CTSession::SetPath() may be used, before creating and/or logging on to the session. CTSession::GetPath() may be used to retrieve the path for an active session.

Using this property, it is even possible to have multiple session dictionaries, in separate directories. This is not required, since the concepts of restricting access to different users to diverse tables or databases may be implemented inside a unique session dictionary.

The example below shows how to use the CTSession::SetPath() function to create a session dictionary file in a user specified directory, instead of the default directory:

Example

CTSession ASession;
try
{
    ASession.SetPath("\new\session");
    ASession.Create("FAIRCOMS", "ADMIN", "ADMIN");
} catch (CTException &err)
{
    printf("Create session failed with error %d\n", err.GetErrorCode());
}

Managing Databases
Once the user performs a successful Session logon, the session object can be used to operate on databases.

Every time a new database is created, or an existing database is added to a session, the database properties such as database name and path are placed in an entry of the database dictionary file.

Every time a user activates a database by connecting to it using CTDDatabase::Connect() method, the database is placed in a list of active (connected) databases within the session. When a database is deactivated or disconnected by CTDDatabase::Disconnect() method, the database is removed from the list of active databases. A user may query the active database list by locating the first active database, the next active database or finding a specific active database.
Creating a new Database

Use `CTSession::CreateDatabase()` method to create a new database. `CTSession::CreateDatabase()` takes a database name and the path where the database dictionary file is to be created. If the database path is NULL or empty (""") the database is created in the server directory, for client/server applications, or in the current directory for standalone applications.

```cpp
// create a new database MyDatabase
try {
    ASession.CreateDatabase("MyDatabase", "");
} catch (CTException &err) {
    printf("Create database failed with error %d\n", err.GetErrorCode());
}
```

`CTSession::CreateDatabase()` creates a new database dictionary file with the database name and extension .FDD (FairCom Database Dictionary). Using the example above, the database dictionary file created is `MyDatabase.fdd`.

Adding an existing Database

An existing database may be added or imported to a session by calling the `CTSession::AddDatabase()` method. Use `CTSession::AddDatabase()` to add an existing database to the current session. `CTSession::AddDatabase()` takes a database name and the path where the database is located.

```cpp
// add MyDatabase to the current session
try {
    ASession.AddDatabase("MyDatabase", "");
} catch (CTException &err) {
    printf("Add database failed with error %d\n", err.GetErrorCode());
}
```

Dropping a Database

When you drop a database from a session, the database information is removed from the session dictionary, but the database dictionary file is left untouched. The drop database operation can be reversed with an add database operation. Drop a database from a session by calling `CTSession::DropDatabase()` method.

```cpp
// drop MyDatabase from current session
try {
    ASession.DropDatabase("MyDatabase");
} catch (CTException &err) {
    printf("Drop database failed with error %d\n", err.GetErrorCode());
}
Deleting a Database

When you delete a database from a session, the database information is removed from the session dictionary and the database dictionary file is deleted. The delete database operation cannot be reversed and the database dictionary data will be lost. Delete a database from a session by calling `CTSession::DeleteDatabase()` method.

```cpp
// delete MyDatabase from current session
try {
    ASession.DeleteDatabase("MyDatabase");
} catch (CTException &err) {
    printf("Delete database failed with error %d\n", err.GetErrorCode());
}
```

First Database

`CTSession::FirstDatabase()` will retrieve the name and path of the first database in a session. If the session has no databases, `CTSession::FirstDatabase()` returns NO (false).

Next Database

`CTSession::NextDatabase()` will retrieve the name and path of the next database in a session. `CTSession::NextDatabase()` returns NO (false) when no more databases exist for the current session.

```cpp
// display all databases in a session
CTDBRET DisplayDatabases(CTSession& ASession) {
    CTDBRET Retval = CTDBRET_OK;
    try {
        CTString dbName;
        CTString dbPath;
        if (ASession.FirstDatabase(dbName, dbPath))
            do {
                printf("Database: %s Path: %s\n", dbName.c_str(), dbPath.c_str());
            } while (ASession.NextDatabase(dbName, dbPath);
    } catch (CTException &err) {
        Retval = err.GetErrorCode();
    }
    return Retval;
}
```

Find Database

`CTSession::FindDatabase()` locates a specific database given the database name and, if the database exists, retrieves the database path. If a database cannot be found, `CTSession::FindDatabase()` returns NO (false).

```cpp
// return YES if database exist or NO if database does not exit
CTBOOL DatabaseExist(CTSession& ASession, CTString& dbName) {
```
CTBOOL Retval;
CTString dbPath;

try
{
    Retval = ASession.FindDatabase(dbName, dbPath);
} catch (CTException &err) {
    Retval = NO;
} return Retval;

First Active Database

CTSession::GetFirstActive() retrieves a database object pointer of the first active database. If
the session contains no active databases, CTSession::GetFirstActive() returns NULL.

Next Active Database

CTSession::GetNextActive() retrieves a database object pointer of the next active database.
When no more active database exist, CTSession::GetNextActive() returns NULL.

// Display all active databases
void DisplayActiveDatabases(CTSession &ASession) {
    VRLEN hScan;
    CTDatabase* pDatabase;

    if ((pDatabase = ASession.GetFirstActive(&hScan)) != NULL) {
        do
        {
            printf("Database: %s Path: %s\n", pDatabase->GetName().c_str(),
                   pDatabase->GetPath().c_str());
            hDatabase = ASession.GetNextActive(&hScan);
        }
        while (pDatabase != NULL);
    }
}

Find Active Database

CTSession::FindActive() locates a specific active database and returns the database object
pointer. If the database is not active, CTSession::FindActive() returns NULL.

// Check if database is active
CTBOOL IsDatabaseActive(CTSession &ASession, const CTString& dbName) {
    return (ASession.FindActive(dbName) != NULL) ? YES : NO;
}

The function above is shown for example purposes only as the c-treeDB API method
CTDatabase::IsActive() provides a more efficient way to check if a database is active or not.
Database UID (Unique IDentifier)

When a database is created or added to a session, an automatic and unique identifier (UID) is associated with the database. A database UID is unique within the session.

A database UID is an unsigned long value that can be used as an alternative method to operate on databases, once the database is created or added to the session.

Find Database by UID

The overloaded method `CTSession::FindDatabase()` locates a database given the database UID and retrieves the database name and path. The following example shows how to implement a database connect procedure using the database UID instead of the database name.

```c++
// Database Connect using UID
CTDatabase* ConnectByUID(CTSession &ASession, ULONG uid)
{
    CTString dbName;
    CTString dbPath;
    CTDatabase *Retval = NULL;

    if (ASession.FindDatabase(uid, dbName, dbPath)
    {
        Retval = new CTDatabase(ASession);
        if (Retval)
        {
            Retval->Connect(dbName);
        }
        else
        {
            throw CTException(CTDBRET_NOMEMORY);
        }
    }
    else
    {
        throw CTException(FNOP_ERR);
    }
    return Retval;
}
```

Find active Database by UID

The overloaded `CTSession::FindActive()` locates an active database given its UID number and returns the database object pointer. The following example shows how to check if a database is active using its UID number.

```c++
// check if database is active, by UID
CTBOOL IsActiveDatabase(CTSession &ASession, ULONG uid)
{
    return (ASession.FindActive(uid) != NULL) ? YES : NO;
}
```

Working with Sessions without Dictionary Support

There may be situations where it is necessary to operate with a table that does not belong to a database, or where a session or database dictionary file is not required. For these situations the user may want to create a session object of type `CTSESSION_CTREE`.

```c++
// create a session object without dictionary support
```
CTSession ASession(CTSESSION_CTREE);
// Logon to a session
try
{
    ASession.Logon("FAIRCOMS", "ADMIN", "ADMIN");
} catch (CTException &err)
{
    printf("Session logon failed with code %d\n", err.GetErrorCode());
}

The **CTSession::Logon()** call above will perform a c-tree logon but it will not attempt to open a session dictionary file. The same result is obtained if a normal **CTSESSION_CTDB** or **CTSESSION_SQL** session is allocated but the method **CTSession::SetLogonOnly()** is called before **CTSession::Logon()**.

// create a session object without dictionary support
CTSession ASession;
// Logon to a session
try
{
    ASession.SetLogonOnly();
    ASession.Logon("FAIRCOMS", "ADMIN", "ADMIN");
} catch (CTException &err)
{
    printf("Session logon failed with code %d\n", err.GetErrorCode());
}

**Attach and Detach Existing Sessions**

There are situations where an existing connection to c-tree already exists, via a call to a low level or ISAM c-tree initialization function, but c-treeDB functionality is required without terminating the existing connection and starting a new c-treeDB session. The following methods are available to permit a session handle to be attached and detached from an existing c-tree connection.

**CTSession::Attach()** attaches an inactive session handle to an existing c-tree Plus or c-treeDB session. Attached sessions have no information on server, user name or user password, and these values are set to NULL. If a **Logout()** is performed on an attached session handle, no session logout is performed and a **DetachSession()** call is executed instead.

There are three valid **mode** values:

<table>
<thead>
<tr>
<th>mode Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTATTACH_SESSION</td>
<td>Attach to a c-treeDB session whose session handle is pointed by parameter source.</td>
</tr>
<tr>
<td>CTATTACH_CTREEID</td>
<td>Attach to a c-tree Plus session whose instance id string is pointed by parameter source.</td>
</tr>
<tr>
<td>CTATTACH_CURRENT</td>
<td>Attach to the current c-tree instance. Contents of parameter source is ignored, as the c-tree instance id is obtained by calling <strong>WCHCTREE()</strong> function.</td>
</tr>
</tbody>
</table>

**CTSession::Detach()** detaches a c-treeDB session handle. The c-tree ISAM or low-level un-initialization is not called, but the session handle control structures are released and
re-initialized. \textit{Handle} is a session handle allocated by \texttt{AllocSession()}. \texttt{DetachSession()} returns CTDBRET_OK on success.

# Session Dictionary

The session dictionary is a collection of databases, and each record of the session dictionary file contains the information of a particular database. In general, just one session dictionary file exists, \texttt{ctdbdict.fsd}, located by default in the server directory (client/server development) or in the application directory (stand-alone application). The table below represents the general layout of a session dictionary file:

### Session Dictionary Layout

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>\texttt{CT_INT4}</td>
<td>4</td>
<td>Record type: 1-database 2-table 3-index</td>
</tr>
<tr>
<td>STATUS</td>
<td>\texttt{CT_INT4}</td>
<td>4</td>
<td>Record status: 0-status ok 1-reserved</td>
</tr>
<tr>
<td>NAME</td>
<td>\texttt{CT_FSTRING}</td>
<td>128</td>
<td>Name of database or table or index</td>
</tr>
<tr>
<td>LINK</td>
<td>\texttt{CT_FSTRING}</td>
<td>128</td>
<td>Name of table if record type is 3-index</td>
</tr>
<tr>
<td>LINKNBR</td>
<td>\texttt{CT_INT4}</td>
<td>4</td>
<td>Table UID if record type is 3-index</td>
</tr>
<tr>
<td>PATH</td>
<td>\texttt{CT_FSTRING}</td>
<td>256</td>
<td>Path of database dictionary, data or index</td>
</tr>
<tr>
<td>SUPEXT</td>
<td>\texttt{CT_FSTRING}</td>
<td>16</td>
<td>Super file extension (if super file is used)</td>
</tr>
<tr>
<td>DATEXT</td>
<td>\texttt{CT_FSTRING}</td>
<td>16</td>
<td>Data file extension (usually .DAT)</td>
</tr>
<tr>
<td>IDXEXT</td>
<td>\texttt{CT_FSTRING}</td>
<td>16</td>
<td>Index file extension (usually .IDX)</td>
</tr>
<tr>
<td>VERSION</td>
<td>\texttt{CT_INT4}</td>
<td>4</td>
<td>Record version: 0x00010000 (version 1.0)</td>
</tr>
<tr>
<td>COUNTER</td>
<td>\texttt{CT_INT4}</td>
<td>4</td>
<td>Deprecated.</td>
</tr>
<tr>
<td>UID</td>
<td>\texttt{CT_INT4}</td>
<td>4</td>
<td>UID for database or table or index</td>
</tr>
<tr>
<td>OWNER</td>
<td>\texttt{CT_FSTRING}</td>
<td>128</td>
<td>Name of owner - used by the c-treeSQL server</td>
</tr>
<tr>
<td>MIRROR_NAME</td>
<td>\texttt{CT_FSTRING}</td>
<td>128</td>
<td>Name of mirrored file.</td>
</tr>
<tr>
<td>MIRROR_PATH</td>
<td>\texttt{CT_FSTRING}</td>
<td>128</td>
<td>Path of mirrored file.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>\texttt{CT_ARRAY}</td>
<td>3128</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>

This information is included for general information on the session dictionary structure. c-treeDB has appropriate functions to retrieve any needed information from the session dictionary, such as the database UID or path.

A session dictionary file has several different record types:

- Database records (the field Type set to 1) holds information about databases.
- A special type four record which maintained the COUNTER field has been been deprecated as of c-treeACE V9.0.
3.3 Working with Databases

A database is a collection of tables and a session may contain several different databases. A database object is required before any table or data operations may take place. The following are typical operations performed on a database:

- Create a database object
- Connect to a database by calling `CTDatabase::Connect()`
- Perform table, index, field and record operations
- When you are done with the database, disconnect by calling `CTDatabase::Disconnect()`
- Destroy the database object if it was created with the new operator

Creating a Database object

A valid database object is required to perform any database operation. You need to pass a valid `CTSession` object to the `CTDatabase` constructor.

```cpp
// create a CTDatabase object
CTDatabase ADatabase(ASession);
try
{
    // connect to database "MyDatabase"
    ADatabase.Connect("MyDatabase");
}
catch (CTException &err)
{
    printf("Database connect failed with error %d\n", err.GetErrorCode);
}
```

If you create a dynamic `CTDatabase` object with the new operator, you are required to destroy the object with the delete operator.

```cpp
// create a dynamic CTDatabase object
CTDatabase* pDatabase = new CTDatabase(ASession);
if (!pDatabase)
{
    printf("CTDatabase creating failed\n");
} 
... other operations ..
// destroy the CTDatabase object
delete pDatabase;
```

If you destroy an active database handle, all open tables associated with the database will be closed and the database will be disconnected from the session.

Connecting to a database

Before performing any operations with a database, a database object must be connected to a session by calling `CTDatabase::Connect()` method. The database should already have been created or added to the session.

```cpp
// connect to a database
CTDatabase ADatabase(ASession);
try
{
    // ...
}
```
{  
ADatabase.Connect("MyDatabase");
}
catch (CTException &err)  
{
    printf("Connect to database failed with error %d\n", err.GetErrorCode());
}

When a database is connected to the session, it is considered "active". Use
CTDatabase::IsActive() to check if a particular database is active.

When the database connection is no longer needed, it must be disconnected from the session by
calling CTDatabase::Disconnect() method.

    // disconnect from a database  
try
    {
        ADatabase.Disconnect();
    }  
catch (CTException &err)
    {
        printf("Disconnect from database failed with error %d\n", err.GetErrorCode());
    }

Database properties

The default database properties are suitable for most c-treeDB applications. Advanced
developers may need to tune c-treeDB to meet application requirements.

Database name

Once the database object connects, retrieve the database name with CTDatabase::GetName().

    // display the database name
printf("Database: %s\n", ADatabase.GetName().c_str());

Database path

The database path, by default, is the server directory for client/server applications, or the
application directory for non-server applications. To set a different database path, when the
database is being created, just insert the appropriate path as the second parameter of
CTSession::CreateDatabase(). With this information stored in the Session dictionary, the user
does not need to know where it is located, since it will be automatically retrieved given the
database name.

Once the database is successfully connected to a session, obtain the database path with
CTDatabase::GetPath().

    // display database properties
void DisplayDatabaseProperties(CTDatabase &ADatabase)
    {
        printf("Database: %s\n", ADatabase.GetName().c_str());
        printf("Path: %s\n", ADatabase.GetPath().c_str());
        printf("Number of tables: %d\n", ADatabase.GetTableCount());
    }
Table count

CTDatabase::GetTableCount() retrieves the number of tables associated with a database. CTDatabase::GetTableCount() return -1 (minus 1) if the database is not connected or if the database object is invalid.

```cpp
// display the number of tables in database
printf("Number of tables: %d\n", ADatabase.GetTableCount());
```

Managing Tables

Once the user performs a successful database connect, the database handle can be used to operate on tables.

Every time a new table is created, or an existing table is added to a database, some of the table properties such as table name, path, data file extension, index file extension, etc, are placed in an entry of the database dictionary file.

Every time a user activates a table by opening it with CTTable::Open() method, the table is placed in a list of active (opened) tables within the database. When a table is deactivated, or closed by calling CTTable::Close() method, the table is removed from the list of active tables in the database. A user may query the active table list by locating the first active table, the next active table and finding a specific active table.

Adding an existing table

An existing table may be added or imported to a database by calling the CTDatabase::AddTable() method. CTDatabase::AddTable() takes as parameters the table name and the table path.

```cpp
// add MyTable to the current database
try
{
    ADatabase.AddTable("MyTable", "");
}
catch (CTException &err)
{
    printf("Add table failed with error %d\n", err.GetErrorCode());
}
```

Adding an existing table under transaction control

An extra level of data integrity can be achieved when you add an existing table to a database under transaction control. When the transaction is committed, the database dictionary data for the table is committed to disk. If the transaction is aborted, the dictionary data for the table is automatically removed from the database dictionary.

The code fragment below shows how to add an existing table under transaction control.

```cpp
// begin a transaction
ADatabase.Begin();
try
{
    // add MyTable to the current database
    ADatabase.AddTable("MyTable", "");
    // commit the transaction
    ADatabase.Commit();
}
```
catch (CTException &err)
{
    // abort the transaction
    ADatabase.Abort();
    printf("Add table failed with code %d\n", err.GetErrorCode());
}

Dropping a table

When you drop a table from a database, the table information is removed from the database dictionary, but the table data and index files are left untouched. The drop table operation can be reversed with an add table operation. Drop a table from a database by calling CTDatabase::DropTable() method.

// drop MyTable from current database
try
{
    ADatabase.DropTable("MyTable");
} catch (CTException &err)
{
    printf("Drop table failed with code %d\n", err.GetErrorCode());
}

Dropping a table under transaction control

An extra level of data integrity can be achieved when you drop a table from a database under transaction control. When the transaction is committed, the changes to the database dictionary data for the table are committed to disk. If the transaction is aborted, the dictionary data for the table is automatically restored to the database dictionary.

The code fragment below shows how to drop an existing table under transaction control. No error checking is included in the sample code:

// start a transaction
ADatabase.Begin();
try
{
    // drop MyTable from current database
    ADatabase.DropTable("MyTable");
    // commit the transaction
    ADatabase.Commit();
} catch (CTException &err)
{
    // abort the transaction
    ADatabase.Abort();
    printf("Drop table failed with code %d\n", err.GetErrorCode());
}

Deleting a table

When you delete a table from a database, the table information is removed from the database dictionary and the table data and index files are deleted from disk. The delete table operation can be reversed only when used under transaction control. Without transaction control, a delete table operation will delete the data and index files and the table data will be lost. Delete a table from a database by calling CTDatabase::DeleteTable() method. Example:

// delete MyTable from current database
try
Deleting a table under transaction control

An extra level of data integrity can be achieved when you delete a table from a database under transaction control. When the transaction is committed, the changes to the database dictionary data for the table are committed to disk and the table and index files are deleted from disk. If the transaction is aborted, the dictionary data for the table is automatically restored to the database dictionary and the original data and index files are restored to their original state.

The code fragment below shows how to delete an existing table under transaction control. No error checking is included in the sample code:

```cpp
// start a transaction
try
{
    // delete MyTable from current database
    CTString password;
    ADatabase.DeleteTable("MyTable", password);

    // commit the transaction
    ADatabase.Commit();
} catch (CTException &err) {
    // abort the transaction
    ADatabase.Abort();
    printf("Delete table failed with code %d\n", err.GetErrorCode());
}
```

**Note:** The `DeleteTable()` method takes as parameters the table name and the table password. Set the password parameter to an empty `CTString` object if a table was created without passwords.

First Table

`CTDatabase::FirstTable()` retrieves the name and path of the first table in a session. If the session has no tables, `CTDatabase::FirstTable()` returns NO (false). See the example in "Next Table" (page 81).

Next Table

`CTDatabase::NextTable()` retrieves the name and path of the next table in a database. `CTDatabase::NextTable()` returns NO (false) when no more tables exist for the current database.

```cpp
// Display all tables in a database
void DisplayTables(CTDatabase &ADatabase) {
    CTString Name;
    CTString Path;
```
if (ADatabase.FirstTable(Name, Path))
do{
    printf("Table: %s Path: %s\n", Name.c_str(), Path.c_str());
}while (ADatabase.NextTable(Name, Path));

Find Table

CTDatabase::FindTable() locates a specific table given the table name and, if the table exists, retrieves the table path. If a table cannot be found, CTDatabase::FindTable() returns NO (false).

// return YES if table exist or NO if table does not exit
CTBOOL TableExist(CTDatabase &ADatabase, CTString& tblName)
{
    CTString tblPath;
    return ADatabase.FindTable(TblName, tblPath);
}

First Active Table

CTDatabase::GetFirstActive() retrieves the table object pointer of the first active table. If the database contains no active tables, CTDatabase::GetFirstActive() returns NULL.

Next Active Table

CTDatabase::GetNextActive() retrieves the table object pointer of the next active table. When no more active tables exist, CTDatabase::GetNextActive() returns NULL.

// Display all active tables
void DisplayActiveTables(CTDatabase &ADatabase)
{
    VRLEN hScan;
    CTTable *pTable;
    if ((pTable = ADatabase.GetFirstActive(&hScan)) != NULL)
    {
        do
        {
            printf("Table: %s Path: %s\n", pTable->GetName().c_str(),
            pTable->GetPath().c_str());
            pTable = ADatabase.GetNextActive(&hScan);
        }while (pTable != NULL);
    }
}

Find Active Table

CTDatabase::FindActive() locates a specific active table and returns the table object pointer. If the table is not active, CTDatabase::FindActive() returns NULL.

// Check if table is active
CTBOOL IsTableActive(CTDatabase& ADatabase, CTString& tblName)
{
    return (ADatabase.FindActive(tblName) != NULL) ? YES : NO;
}
The function above is shown for example purposes only as the c-treeDB method `CTTable::IsActive()` provides a more efficient way to check if a table is active.

**Table UID (Unique IDentifier)**

When a table is created or added to a database, an automatic and unique identifier (UID) is associated with the table. A table UID is unique within the database that the table is associated with.

A table UID is an unsigned long value that can be used as an alternative method to operate on tables once the table is created or added to a database.

**Find Table by UID**

The overloaded method `CTDatabase::FindTable()` locates a table in the database given the table UID and retrieves the table name and path. The following example shows how to implement a table open function using the table UID instead of the table name.

```c++
// open table using UID
CTTable* OpenByUID(CTDatabase& ADatabase, ULONG uid, CTOPEN_MODE OpenMode)
{
    CTString tblName;
    CTString tblPath;
    CTTable* Retval;

    // locate the table in the database by uid
    if (ADatabase.FindTable(uid, tblName, tblPath))
    {
        Retval = new CTTable(ADatabase);
        if (!Retval)
            throw CTException(CTDBRET_NOMEMORY);
        try
        {
            Retval->Open(tblName, OpenMode);
        } catch (CTException& err)
        {
            delete Retval;
            throw;
        }
    }
    else
    {
        // table not found
        throw CTException(INOT_ERR);
    }
    return Retval;
}
```

**Find active Table by UID**

The overloaded method `CTDatabase::FindActive()` locates an active table given its UID number and returns the active table object pointer. The following example shows how to check if a table is active using its UID number.

```c++
// check if a table is active, by UID
CTBOOL IsTableActive(CTDatabase& ADatabase, ULONG uid)
```
{ return (ADatabase.FindActive(uid) != NULL) ? YES : NO; }

Database Dictionary

A database must have a database dictionary; this file has the database name with extension .fdd. It is created with CTDdatabase::CreateDatabase(), and must exist for the user to connect to the database. Each database dictionary maintains information about the tables associated with the database. When a database is connected to the session, it is considered "active".

Database Dictionary Layout

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>CT_INT4</td>
<td>4</td>
<td>Record type: 1-database 2-table 3-index</td>
</tr>
<tr>
<td>STATUS</td>
<td>CT_INT4</td>
<td>4</td>
<td>Record status: 0-status ok 1-reserved</td>
</tr>
<tr>
<td>LOGICAL_NAME</td>
<td>CT_FSTRING</td>
<td>128</td>
<td>Logical (database) name of database or table or index</td>
</tr>
<tr>
<td>PHYSICAL_NAME</td>
<td>CT_FSTRING</td>
<td>128</td>
<td>Physical name of database or table or index</td>
</tr>
<tr>
<td>LINK</td>
<td>CT_FSTRING</td>
<td>128</td>
<td>Name of table if record type is 3-index</td>
</tr>
<tr>
<td>LINKNBR</td>
<td>CT_INT4</td>
<td>4</td>
<td>Table UID if record type is 3-index</td>
</tr>
<tr>
<td>PATH</td>
<td>CT_FSTRING</td>
<td>256</td>
<td>Path of database dictionary, data or index</td>
</tr>
<tr>
<td>SUPEXT</td>
<td>CT_FSTRING</td>
<td>16</td>
<td>Super file extension (if super file is used)</td>
</tr>
<tr>
<td>DATEXT</td>
<td>CT_FSTRING</td>
<td>16</td>
<td>Data file extension (usually .DAT)</td>
</tr>
<tr>
<td>IDXEXT</td>
<td>CT_FSTRING</td>
<td>16</td>
<td>Index file extension (usually .IDX)</td>
</tr>
<tr>
<td>VERSION</td>
<td>CT_INT4</td>
<td>4</td>
<td>Record version: 0x00010000 (version 1.0)</td>
</tr>
<tr>
<td>COUNTER</td>
<td>CT_INT4</td>
<td>4</td>
<td>Deprecated.</td>
</tr>
<tr>
<td>UID</td>
<td>CT_INT4</td>
<td>4</td>
<td>UID for database or table or index</td>
</tr>
<tr>
<td>OWNER</td>
<td>CT_FSTRING</td>
<td>128</td>
<td>Name of owner - used by the c-treeACE SQL server</td>
</tr>
<tr>
<td>MIRROR_NAME</td>
<td>CT_FSTRING</td>
<td>128</td>
<td>Name of mirrored file.</td>
</tr>
<tr>
<td>MIRROR_PATH</td>
<td>CT_FSTRING</td>
<td>128</td>
<td>Path of mirrored file.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>CT_ARRAY</td>
<td>3000</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>

This information is included for general information on the database dictionary structure. c-treeDB has appropriate functions to retrieve any needed information from the database dictionary, such as the table UID or path.

A database dictionary file has several different record types:

- Table records (the field Type set to 2) holds information about a data file (table).
- Index records (the field Type set to 3) holds information about an index file.
- A special type four record which maintained the COUNTER field has been been deprecated as of c-treeACE V9.0.
3.4 Working with Tables

A database may contain multiple tables, and a table may belong to multiple databases. Tables created using the c-treeDB interface are kept as files with the extension `.dat`. Indices are stored in separate files with the extension `.idx`. The general process to create and use a c-treeDB table and its associated indices is as follows:

- Create a table object
- Add or insert fields
- Add indices
- Add index segments
- Create the table
- Open the table
- Operate on records

Before you can work with an existing table, the table must be opened as follows:

- Create a table object
- Open the table
- Operate on records

With c-treeDB, it is possible to modify an existing table. The general process to modify a table after it has been created is as follows:

- Open a table
- Make changes to:
  - Fields and/or
  - Indices and/or
  - Segments
- Call the alter table method `CTable::Alter()`.

More details on this process are described in “Altering a table” (page 99).

In general, the table creation is done once in the application, the table is opened, and the data is added in a separate routine. During table creation, besides the table layer itself, three c-treeDB layers will be directly involved: the fields, indices and segments. These layers are only directly involved when the table is being created, or in the event the table structure is modified. When records are being added to or searched in the table, these layers are not relevant, but the record layer is crucial.

With this in mind, this section on Tables discusses these layers in the appropriate order:

- Table creation using fields, indices, and segments.
- Table parameters and properties.
- Opening and manipulating tables.
Creating a Table object

A valid table object is required before most table operations can take place. You need to pass a valid `CTDatabase` object to the `CTTable` constructor.

```
// create a CTTable object
CTTable ATable(ADatabase);
try
{
// open table "MyTable"
ATable.Open("MyTable", CTOPEN_NORMAL);
}
catch (CTException &err)
{
printf("Table open failed with error %d\n", err.GetErrorCode);
}
```

If you create a dynamic `CTTable` object with the new operator, you are required to destroy the object with the delete operator.

```
// create a dynamic CTTable object
CTTable* pTable = new CTTable(ADatabase);
if (!pTable)
{
    printf("CTTable creation failed\n");
}
... other operations ..
// destroy the CTTable object
delete pTable;
```

If you destroy an active table object, all record objects associated with the table will be reset and the table closed before the object is destroyed.

Creating a table object without database support

It is possible to create or open a table without database support by passing a session object when creating the table object. Start your session with the `CTSESSION_CTREE` mode.

```
// create the session and table objects
CTSession ASession(CTSESSION_CTREE);
CTTable ATable(ASession);

// logon to session
ASession.Logon("FAIRCOMS", "ADMIN", "ADMIN");
// without database support, it is necessary to
// specify the path where the table is located
ATable.SetPath("C:\MyDocuments");
// open the table
try
{
    ATable.Open("MyTable", CTOPEN_NORMAL);
}
catch (CTException &err)
{
    printf("Table open failed with error %d\n", err.GetErrorCode());
}
```

Please note from the code above the need to specify the path where the table is located. If no path is specified, c-treeDB will try to open the table from the current directory. The same principle applies when creating a table without database support:
// create the session and table objects
CTSession ASession(CTSESSION_CTREE);
CTTable ATable(ASession);

// logon to session
ASession.Logon("FAIRCOMS", "ADMIN", "ADMIN");
// add fields to table
ATable.AddField("Field1", CT_INT2, 2);
ATable.AddField("Field2", CT_FSTRING, 30);
// without database support, it is necessary to
// specify the path were the table is located
ATable.SetPath("C:\MyDocuments");
// open the table
try
{
    ATable.Create("MyTable", CTCREATE_NORMAL);
}
catch (CTException &err)
{
    printf("Create open failed with error %d
n, err.GetErrorCode());
}

Creating a New Table

Creating a new table may be one of the most crucial operations performed by a database developer or administrator. The c-treeDB API offers a powerful, yet easy-to-use, mechanism for creating tables.

The create table process does not leave the table open after the table is created. The user must explicitly open the table to be able to add data records and query any of the table properties.

The create table process involves the following steps:

- Allocate a new table handle
- Add, insert, or delete fields
- Optionally, add, insert, or delete indices
- Change any of the default table properties
- Create the table

Adding, inserting or deleting fields

Fields are what hold the actual record data for each row in a table. Whereas a record could be considered a "row" in a table, a field could be considered a "column" in a table, or a "cell" in a record. The fields are defined at the time of the table creation.

Fields are added to the record definition in the order they are declared. The c-treeDB API also includes a set of functions that will allow a field to be inserted at a certain place in the record definition and fields can also be deleted from the record definition.

Each filed will have a file type (e.g., CT_INTEGER and CT_CHAR in the example below). For more information, see Field Types (page 227).

CTTable::AddField() method will add a new field to the end of the record definition.

// create a new table object
CTTable ATable(ADatabase);
// add two fields to the table record definition
ATable.AddField("Field1", CT_INTEGER, 4);
ATable.AddField("Field2", CT_CHAR, 30);
// create the table
ATable.Create("MyTable", CTCREATE_NORMAL);

CTTable::InsField() inserts a new field before a given field that has already been defined.
CTTable::InsField() overloaded methods allow you to specify a field index or a field name to
identify the position on the record definition that the new field will be inserted. The first field is
number 0, the second field is number 1 and so on.

// create a new table object
CTTable ATable(ADatabase);
// add two fields to the table record definition
ATable.AddField( "Field1", CT_INTEGER, 4);
ATable.AddField( "Field2", CT_CHAR, 30);
ATable.InsField("Field2", "Field3", CT_BOOL, 1);
// create the table
ATable.Create("MyTable", CTCREATE_NORMAL);

CTTable::DelField() deletes a field from the record definition. CTTable::DelField() overloaded
methods allow the user to specify a field index or a field name to identify the field to delete. The
first field is number 0, the second field is number 1, and so on.

// create a new table object
CTTable ATable(ADatabase);
// add two fields to the table record definition
ATable.AddField( "Field1", CT_INTEGER, 4);
ATable.AddField( "Field2", CT_CHAR, 30);
// delete field2 from record definition
ATable.DelField("Field2");
// create the table
ATable.Create("MyTable", CTCREATE_NORMAL);

Fixed or variable length records

The c-treeDB API automatically, and transparently, handles the details of fixed and variable
length records. A table is set to variable length if it has at least one variable length field. c-treeDB
scans the user field definitions until it encounters the first variable length field. If a table contains
no variable length fields, the record is set to fixed length.

c-treeDB also automatically calculates the size of the fixed portion of a record by adding the size
of the fixed length fields, taking into consideration the field alignment in the record buffer, until the
first variable length field is encountered. The variable length fields are listed below, with the
matching c-tree Plus data type in parentheses:

CT_PSTRING    (CT_PSTRING)
CT_VARBINARY  (CT_2STRING)
CT_LVB        (CT_4STRING)
CT_VARCHAR    (CT_STRING)
CT_LVC        (CT_STRING)

Any type of field can be placed anywhere in the record buffer and also be used as an index
segment. c-treeDB makes full use of this feature by providing the user with an advanced dynamic
record buffer management.
Hidden fields

There are three special fields that are automatically included by c-treeDB in the table record definition. c-treeACE SQL makes extensive use of the null flag and ROWID fields:

- the delete flag ($DELFLD$)
- the null flag ($NULFLD$)
- the ROWID fields ($ROWID$)

These fields are transparently and automatically handled by the c-treeDB API and can’t be handled directly by the field handling functions of the c-treeDB API. There are specific functions that will, in some cases, retrieve data and status information kept by the hidden fields. These fields are also optional and c-treeDB will operate correctly with tables that do not possess them. There are also create table modes that allow the developer creating c-treeDB tables without any one, or all, of the hidden fields.

The delete flag field is for internal deletion purposes and should not be modified. $DELFLD$ is a CT_ARRAY field of four bytes. The only purpose of this field is to keep a place at the beginning of the record to hold a c-tree delete flag byte when the record is deleted. Four bytes are used instead of just one byte due to alignment requirements. This is an internal c-treeDB requirement, and should not be modified or touched by the user.

$NULFLD$ is a CT_ARRAY field with the size based on the number of user defined fields for the table. For each user defined field, one bit is reserved in $NULFLD$. The $NULFLD$ field keeps the NUL flag information for each user defined field. If a user field is null, the corresponding bit in $NULFLD$ will be set. When data is written to the field, the corresponding bit is cleared. The user should never modify or verify this field directly, but should use the appropriate API functions:

- Use CTRecord::IsNullField() or CTField::GetNullFlag() to verify the if a field contains null data or not.
- Use CTFIELD::SetNullFlag() to set the null flag for a particular field.
- Other functions that clear the null flag are CTRecord::ClearField() and CTRecord::Clear().

$ROWID$ is a CT_INT8 (64-bit integer) field holding the auto increment value generated by c-tree every time a new record is added to the table. This field is a read-only field that acts as a unique record identifier. Retrieve the $ROWID$ using CTRecord::GetRowid(), or locate a record given its rowid using CTRecord::FindRowid(). To find out if a table has support for rowid, use CTTable::HasRowid().

$ROWID$ is used by c-treeACE SQL as a unique record identifier. For ISAM files or c-treeDB tables created with CTCREATE_NOROWID flag the $ROWID$ field will not exist. In this case the RECBYT offset will be used instead.

**Note:** Record offsets may change for a given variable-length record when a record is updated to a larger record size than the original record. Thus the RECBYT index cannot be used as a unique record identifier.

By default, c-treeDB creates the three hidden fields. Tables created with the c-tree Plus ISAM or low-level API will not include these fields by default. c-treeDB does not require the hidden fields to operate, but they allow more advanced capabilities. When creating a new table, users may disable the inclusion of the hidden fields by using the create modes CTCREATE_NONULFLD, CTCREATE_NODELFLD, and CTCREATE_NOROWID.
The default table layout is presented below. Note that the first field added by the user is always field 0.

| $DELFLD$ | $NULFLD$ | $ROWID$ | user field 0 | user field 1 | ... | user field n |

### Adding or deleting indices

Indices and index segments are key-based search tools that make record seeking faster and more efficient. An index is a mapping table that contains keys describing certain records and pointers to those records. An index segment describes the table field from which the keys are created.

Indices are added to the table definition in the order they are declared. The c-treeDB API also includes a set of functions that will allow an index to be deleted from the table index definition.

**CTTable::AddIndex()** will add a new index at the end of the table index definition. For each index added to the table, one or more index segments should also be added to define which field combination form a particular index. **CTTable::AddSegment()** overloaded methods will accomplish the task of adding segments to an index.

```cpp
// create a new table object
CTTable ATable(ADatabase);

// add two fields to the table record definition
ATAble.AddField("Field1", CT_INTEGER, 4);
ATAble.AddField("Field2", CT_CHAR, 30);

// add index 0 - the first index
ATAble.AddIndex("MyIndex1", CTINDEX_FIXED, YES, NO);

// add index 0 segments
ATAble.AddSegment("MyIndex1", "Field1", CTSEG_SCHSEG);

// add index 1 - the second index
ATAble.AddIndex("MyIndex2", CTINDEX_FIXED, NO, NO);

// add index 1 segments
ATAble.AddSegment("MyIndex2", "Field2", CTSEG_SCHSEG);
ATAble.AddSegment("MyIndex2", "Field1", CTSEG_SCHSEG);

// create the table
try
{
    ATable.Create("MyTable", CTCREATE_NORMAL);
}
catch (CTException &err)
{
    printf("Create table failed with error %d\n", err.GetErrorCode());
}
```

The **CTTable::AddIndex()** method takes an index name, index type, and two Boolean flags indicating if the index accepts duplicate keys and if the index should process null keys. The valid index types are:
### c-treeDB Index Type | c-treeDB .NET Index Type | Description
---|---|---
CTINDEX_FIXED | FIXED_INDEX | Fixed-length key
CTINDEX_LEADING | LEADING_INDEX | Fixed-length keys that are likely to have leading character duplication among the key values
CTINDEX_PADDING | PADDING_INDEX | Variable-length keys for which not much leading character duplication is expected.
CTINDEX_LEADPAD | LEADPAD_INDEX | Variable-length keys for which much leading character duplication is expected.
CTINDEX_ERROR | ERROR_INDEX | Index type error.
CTINDEX_DFRIDX | INDEX_DFRIDX | Indicates a deferred index (V11 and later).
CTINDEX_NOMOD | INDEX_NOMOD | Indicates an index with unmodifiable ISAM and c-treeDB keys (V11 and later).

**Note:** c-treeDB .NET Index Key Types are defined in the `KEY_TYPE` enum.

The add and insert segment functions require a segment mode to be passed as the last parameter. Please refer to "Segment Modes" (page 92) describing the valid segment modes.

An index can be deleted from the table index definition by calling one of `CTTable::DelIndex()` overloaded methods.

```cpp
// create a new table object
CTTable ATable(ADatabase);

// add two fields to the table record definition
ATable.AddField("Field1", CT_INTEGER, 4);
ATable.AddField("Field2", CT_CHAR, 30);

// add index 0 - the first index
ATable.AddIndex("MyIndex1", CTINDEX_FIXED, YES, NO);

// add index 0 segments
ATable.AddSegment("MyIndex1", "Field1", CTSEG_SCHSEG);

// add index 1 - the second index
ATable.AddIndex("MyIndex2", CTINDEX_FIXED, NO, NO);

// add index 1 segments
ATable.AddSegment("MyIndex2", "Field2", CTSEG_SCHSEG);
ATable.AddSegment("MyIndex2", "Field1", CTSEG_SCHSEG);

// delete index 0
ATable.DelIndex("MyIndex1");

// create the table
try
{
    ATable.Create("MyTable", CTCREATE_NORMAL);
}
catch (CTException &err)
{
    printf("Create table failed with error %d\n", err.GetErrorCode());
}
```
Segment Modes

The segment modes based on absolute field number, also known as schema fields, are the preferred modes to use in the segment definition. The preferred segment modes are:

- `CTSEG_SCHSEG`
- `CTSEG_USCHSEG`
- `CTSEG_VSCHSEG`
- `CTSEG_UVSCHSEG`
- `CTSEG_SCHSRL`

You may OR in the mode `CTSEG_DESCENDING` to the segment mode to specify the descending sort order for a segment. You can also OR in the segment mode `CTSEG_ALTSEG` to specify an alternate collating sequence for the segment.

Using the preferred segment modes makes c-treeDB based tables fully compatible with ISAM/Low Level applications and/or c-treeACE SQL applications.

<table>
<thead>
<tr>
<th>c-treeDB Segment Modes</th>
<th>c-treeDB .NET Segment Modes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTSEG_SCHSEG</td>
<td>SCHSEG_SEG</td>
<td>Absolute field number</td>
</tr>
<tr>
<td>CTSEG_USCHSEG</td>
<td>USCHSEG_SEG</td>
<td>Absolute field number - uppercase</td>
</tr>
<tr>
<td>CTSEG_VSCHSEG</td>
<td>VSCHSEG_SEG</td>
<td>Absolute field number - pad strings</td>
</tr>
<tr>
<td>CTSEG_UVSCHSEG</td>
<td>UVSCHSEG_SEG</td>
<td>Absolute field number - pad strings upper</td>
</tr>
<tr>
<td>CTSEG_SCHSRL</td>
<td>SCHSRL_SEG</td>
<td>Absolute field number - auto increment</td>
</tr>
<tr>
<td>CTSEG_DESCENDING</td>
<td>DESCENDING_SEG</td>
<td>Descending segment mode</td>
</tr>
<tr>
<td>CTSEG_ALTSEG</td>
<td>ALTSEG_SEG</td>
<td>Alternative collating sequence</td>
</tr>
<tr>
<td>CTSEG_ENDSEG</td>
<td>ENDSEG_SEG</td>
<td>END segment mode</td>
</tr>
</tbody>
</table>

The other segment modes are kept for compatibility with existing c-treeACE applications. Advanced c-treeDB functions like `ctdbAlterTable()` may not work properly if the segment mode is not one of the preferred segment modes.

You may specify these segment modes with `ctdbAddSegmentEx()`, which expects an absolute record offset where the segment is to start instead of a field indicator, the length in bytes of the segment, and the segment mode.

<table>
<thead>
<tr>
<th>c-treeDB Segment Modes</th>
<th>c-treeDB .NET Segment Modes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTSEG_REGSEG</td>
<td>REGSEG_SEG</td>
<td>Absolute byte offset - No transformation</td>
</tr>
<tr>
<td>CTSEG_INTSEG</td>
<td>INTSEG_SEG</td>
<td>Absolute byte offset - unsigned int/long</td>
</tr>
<tr>
<td>CTSEG_UREGSEG</td>
<td>UREGSEG_SEG</td>
<td>Absolute byte offset - uppercase</td>
</tr>
<tr>
<td>CTSEG_SRLSEG</td>
<td>SRLSEG_SEG</td>
<td>Absolute byte offset - auto increment</td>
</tr>
<tr>
<td>CTSEG_VARSEG</td>
<td>VARSEG_SEG</td>
<td>Relative field number</td>
</tr>
</tbody>
</table>
### c-treeDB Segment Modes

<table>
<thead>
<tr>
<th>c-treeDB Segment Modes</th>
<th>c-treeDB .NET Segment Modes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTSEG_UVARSEG</td>
<td>UVARSEG_SEG</td>
<td>Relative field number - uppercase</td>
</tr>
<tr>
<td>CTSEG_SGNSEG</td>
<td>SGNSEG_SEG</td>
<td>Absolute byte offset - signed int/long</td>
</tr>
<tr>
<td>CTSEG_FLTSEG</td>
<td>FLTSEG_SEG</td>
<td>Absolute byte offset - float/double</td>
</tr>
<tr>
<td>CTSEG_DECSEG</td>
<td>DECSEG_SEG</td>
<td>Absolute byte offset - not yet implemented</td>
</tr>
<tr>
<td>CTSEG_BCDSEG</td>
<td>BCDSEG_SEG</td>
<td>Absolute byte offset - not yet implemented</td>
</tr>
<tr>
<td>CTSEG_DESCENDING</td>
<td>DESCENDING_SEG</td>
<td>Descending segment mode</td>
</tr>
<tr>
<td>CTSEG_ALTSEG</td>
<td>ALTSEG_SEG</td>
<td>Alternative collating sequence</td>
</tr>
</tbody>
</table>

c-treeDB .NET Segment Modes are defined in the **SET_MODE** enum.

### ROWID index

Two indices are created by default during the table creation: the **ROWID** and recbyt indices.

The **ROWID** index, formed by a field and an associated index, holds the auto increment value generated automatically by c-tree every time a new record is added to the table.

When a table is created with **ROWID** support, a **ROWID** index is automatically created for the table. The operation of the **ROWID** index is transparent to the c-treeDB user. c-treeDB will automatically update the index entries. The **ROWID** index will not appear in the list of indices for a table. The user cannot alter or delete the components of the **ROWID** index.

Functions that deal with the **ROWID** index are:

- **CTTable::HasRowid()** - to determine if the table has **ROWID** support;
- **CTRecord::GetRowid()** - to retrieve the **ROWID** for a given record;
- **CTRecord::FindRowid()** - to locate and retrieve a record, given the **ROWID**.

By default, all c-treeDB created tables have support for the **ROWID** index. If, for any reason, a table should be created without support for this index, the create mode **CTCREATE_NOROWID** should be added in **CTTable::Create()** method.

### RECBYT index

**RECBYT** indices were introduced in the c-tree Plus ISAM API to provide improved space management for variable length records in a table and to permit backward physical traversal of data files that contain resources and variable length records. A **RECBYT** index is an index based on the byte offset (**recbyt**) of the record being indexed.

**RECBYT** indices are optional and the user can disable their creation by specifying the **CTCREATE_NORECBYT** when a table is created.

When a table is created with **RECBYT** index support, a **RECBYT** index is automatically created for the table. The operations on the **RECBYT** index are transparent to the user. c-treeDB will automatically update the index entries. The **RECBYT** index will not appear in the list of indices for a table. The user cannot alter or delete the components of the **RECBYT** index. The use of
RECBYT index has no impact on the fields in the record buffer of a table. There is no field associated with this index.

Changing default properties

When a table is created, the table properties are set to default values. Developers using the c-treeDB API may need to change the default value of the table properties to suit the design requirements of their applications.

The table property values must be changed after the creating the table object, but before calling CTTable::Create(). The following table properties may be changed:

- Path
- Data file extension
- Index file extension
- Password
- Group ID
- Permission Mask
- Default Data Extent Size
- Default Index Extent Size
- Field padding

Please refer to "Table Properties" (page 107) for more information on the table properties.

Creating the table

After all fields and indices have been defined and the table properties set, it is time to create the table by calling CTTable::Create() method. CTTable::Create() method take as parameters the table name and the create mode.

```cpp
// allocate a new table handle
CTTable ATable(ADatabase);

// add two fields to the table record definition
ATable.AddField("Field1", CT_INTEGER, 4);  // 4 byte integer
ATable.AddField("Field2", CT_CHAR, 30);   // 30 characters

// add index 0 - the first index
ATable.AddIndex("MyIndex1", CTINDEX_FIXED, YES, NO);

// add index 0 segments
ATable.AddSegment("MyIndex1", "Field1", CTSEG_SCHSEG);

// add index 1 - the second index
ATable.AddIndex("MyIndex2", CTINDEX_FIXED, NO, NO);

// add index 1 segments
ATable.AddSegment("MyIndex2", "Field2", CTSEG_SCHSEG);

// create the table
try
{
    ATable.Create("MyTable", CTCREATE_NORMAL);
}
```
catch (CTException &err)
{
  printf("Create table failed with error %d\n", err.GetErrorCode());
}

The table create modes are a combination of the following valid modes. You can specify mode than one create mode by OR-ing the following constants:

<table>
<thead>
<tr>
<th>c-treeDB Table Create Mode</th>
<th>c-treeDB .NET Table Create Mode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTCREATE_NORMAL</td>
<td>NORMAL_CREATE</td>
<td>Normal table creation. Use this mode when no other create mode apply.</td>
</tr>
<tr>
<td>CTCREATE_PREIMG</td>
<td>PREIMG_CREATE</td>
<td>This mode implements transaction processing for a table but does not support automatic file recovery. Files with CTCREATE_PREIMG mode do not take any space in the system transaction logs.</td>
</tr>
<tr>
<td>CTCREATE_TRNLOG</td>
<td>TRNLOG_CREATE</td>
<td>With this mode you will get the full benefit of transaction processing, including both atomicity and automatic recovery. If you are not sure of what mode to use, and you do want to use transaction processing, then use this mode.</td>
</tr>
<tr>
<td>CTCREATE_WRITETHRU</td>
<td>WRITETHRU_CREATE</td>
<td>This mode forces the operating system to flush all disk cache buffers when a data write occurs. Setting this mode can slow performance of the file handler. On the other hand, it is an important feature to use if you want to ensure that all data writes are put to the disk immediately. It is particularly important if you are in an operating environment where the system crashes a lot, and you are not using transactions. However, this mode does not guarantee that operating system buffers will be flushed as expected.</td>
</tr>
<tr>
<td>CTCREATE_CHECKLOCK</td>
<td>CHECKLOCK_CREATE</td>
<td>Tables created with this mode requires a record lock before a record can be updated. If a lock is not obtained, the error code DADV_ERR is returned.</td>
</tr>
<tr>
<td>CTCREATE_NORECBYTY</td>
<td>NORECBYTY_CREATE</td>
<td>Create the table without the RECBYTY index.</td>
</tr>
<tr>
<td>CTCREATE_NOROWID</td>
<td>NOROWID_CREATE</td>
<td>Create the table without the ROWID index.</td>
</tr>
<tr>
<td>CTCREATE_CHECKREAD</td>
<td>CHECKREAD_CREATE</td>
<td>Tables create with this mode requires a record lock as records are read. Obtain at least a read lock on a record before it can be read, otherwise the function will return error code DADV_ERR.</td>
</tr>
<tr>
<td>CTCREATE_HUGEFILE</td>
<td>HUGEFILE_CREATE</td>
<td>Create the table with huge file support. With this mode on, tables will support 8 byte addresses for file offsets.</td>
</tr>
<tr>
<td>CTCREATE_NODELFLD</td>
<td>NODELFLD_CREATE</td>
<td>This mode indicate that the create is to be created without the $DELFLD$ field support.</td>
</tr>
</tbody>
</table>
### Creating a table under transaction control

You can add an extra level of data integrity when creating a new table by placing the code to create a table inside a transaction. If the transaction is aborted, the table entry in the database dictionary file is removed, and the table data and index files are deleted from disk.

When a table is created inside a transaction, and until the transaction is committed or aborted, the newly created table must be open with `CTOPEN_EXCLUSIVE` mode, otherwise the table open operation will fail. After the transaction is committed the table can be open in non exclusive mode.

The code fragment below creates a new table under transaction control. Again no error checking code is included in the example:

```cpp
// create a new table object
CTTable ATable(ADatabase);

// begin a transaction
ATable.Begin();
try{
    // add a field
    ATable.AddField("Field1", CT_INTEGER, 4);

    // add another field
    ATable.AddField("Field1", CT_CHAR, 30);

    // create the table
    ATable.Create("MyTable", CTCREATE_NORMAL);

    // commit the transaction
    ATable.Commit();
}
catch (CTException &err)
{
    // abort the transaction
    ATable.Abort();
    printf("Create table failed with error %d\n", err.GetErrorCode());
}
```

**Note:** It is important to note that if you open a table that was created inside the current transaction, i.e. the transaction has not yet been committed or aborted, you must open the table in exclusive mode by specifying the `CTOPEN_EXCLUSIVE` mode to `CTTable::Open()`.
Opening a table

A table must be opened before any data operations within it can take place. Use the `CTTable::Open()` method to open a table.

```cpp
// open a table
CTTable ATable(ADatabase);
try {
    ATable.Open("MyTable", CTOPEN_NORMAL);
} catch (CTException &err) {
    printf("Open table failed with error %d\n", err.GetErrorCode());
}
```

After opening the table, usual operations like add, update, delete, and search for records can be done. Record operations are described in detail in "Working with Records" (page 113).

The `CTTable::Open()` method take as parameters a table name and the table open mode.

<table>
<thead>
<tr>
<th>c-treeDB File Open Mode</th>
<th>c-treeDB .NET File Open Mode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTOPEN_NORMAL</td>
<td>NORMAL_OPEN</td>
<td>Use this mode if no other open modes apply.</td>
</tr>
</tbody>
</table>
| CTOPEN_DATAONLY         | DATAONLY_OPEN                 | Open only the data table. Used to rebuild a table that may or may not be missing indices.  
  - **Caution:** Updates made to a data file with this file mode will not have any necessary updates reflected in the associated index files. |
<p>| CTOPEN_EXCLUSIVE        | EXCLUSIVE_OPEN                | This mode opens the table as exclusive. If this mode is used, only one user can open a table. If an application already has the file open in any mode, no other application can open the table as CTOPEN_EXCLUSIVE. Once an application opens a table as CTOPEN_EXCLUSIVE, no other application can open it. Reads and writes are cached for index files opened with this file mode since there are no integrity issues with only one process in the file. |</p>
<table>
<thead>
<tr>
<th>c-treeDB File Open Mode</th>
<th>c-treeDB .NET File Open Mode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTOPEN_PERMANENT</td>
<td>PERMANENT_OPEN</td>
<td>Many operating systems and/or C compiler run-time libraries limit the number of files that can be opened at one time. A permanent file open causes the file to be opened and stay open until the program executes a file close. A non-permanent file open causes the table data and index files to be opened, but allows them to be transparently closed and reopened to allow other data and index files to be used. When it is necessary for a data and index file to be temporarily closed, c-treeACE selects the least recently used file. This file remains closed until it is used, at which time it will be automatically reopened. This strategy causes c-treeACE to use all available file descriptors.</td>
</tr>
<tr>
<td>CTOPEN_CORRUPT</td>
<td>CORRUPT_OPEN</td>
<td>This mode opens tables with corrupted indices or in certain cases, tables with corrupted data. With c-treeDB this mode is usually used in conjunction with <code>ctdbAlterTable()</code> mode to perform a rebuild if the indices became corrupted: open table with <code>CTOPEN_CORRUPT</code> mode, then call <code>ctdbAlterTable()</code> with <code>CTDB_ALTER_INDEX</code> mode to force the rebuild of all indices of the table. You can also specify <code>ctdbAlterTable()</code> mode (<code>CTDB_ALTER_INDEX</code> <code>CTDB_ALTER_PURGEDUP</code>) to purge any duplicate records that may cause the index rebuild to fail. If a table table becomes corrupt, the table may be open with <code>CTOPEN_CORRUPT</code> mode and then <code>ctdbAlterTable()</code> with <code>CTDB_ALL_FULL</code> is invoked to try to recover the table.</td>
</tr>
<tr>
<td>CTOPEN_CHECKLOCK</td>
<td>CHECKLOCK_OPEN</td>
<td>Tables opened with this mode requires a record lock before a record can be updated. If a lock is not obtained, the error code DADV_ERR is returned.</td>
</tr>
<tr>
<td>CTOPEN_CHECKREAD</td>
<td>CHECKREAD_OPEN</td>
<td>Tables opened with this mode requires a record lock as records are read. Obtain at least a read lock on a record before it can be read, otherwise the function will return error code DADV_ERR.</td>
</tr>
<tr>
<td>CTOPEN_READONLY</td>
<td>READONLY_OPEN</td>
<td>Opens the table in <code>READONLY</code> mode and does not allow any modifications to the table structure or data records.</td>
</tr>
</tbody>
</table>

**Note:** c-treeDB .NET users can find the open modes listed in the `OPEN_MODE` enum.
Opening a table with password

If a table was created with a password, every time that table is open, we need to specify the correct password for the open table operation to succeed. After the table object is created, but before the table is opened, the table password property must be set.

```c++
// opening a table with password
CTTable ATable(ADatabase);

// set the table password
ATable.SetPassword("MyPassword");

// open the table
try
{
    ATable.Open("MyTable", CTOPEN_NORMAL);
} catch (CTException &err) { printf("Open table failed with error %d\n", err.GetErrorCode()); }
```

Closing a table

After a successful open table, and if the table object is no longer needed, the table should be closed to allow all c-treeDB, c-tree Plus and operating systems buffers to be flushed to disk. It is very good programming practice to always close every open table before the process or thread is terminated.

```c++
// close the table
try
{
    ATable.Close();
} catch (CTException &err) { printf("Close table failed with error %s\n", err.GetErrorCode()); }
```

Altering a table

The c-treeDB alter table function was designed and implemented to allow the modification of table, field and index properties after a table was created, and possibly already populated with data.

The usual steps to perform an alter table are:

- Add, insert, delete or edit fields
- Add, edit or delete indices
- Alter the table by calling `CTTable::Alter()` method

Add, insert, delete, or edit fields

By calling one of the following edit management methods, the table definition is marked as modified. For the changes to be reflected on the data and index files, you must call `ctdbAlterTable()`.
To add, insert or delete a field, call `CTTable::AddField()`, `CTTable::InsField()`, `CTTable::DelField()`, `CTField::SetName()`, `CTField::SetType()`, `CTField::SetLength()`, `CTField::SetPrecision()`, `CTField::setScale()`, and `CTField::SetNullFlag()`.

Most changes relating to fields will trigger the `CTTable::Alter()` to perform a full table build.

**Add, edit or delete indices**

By calling one of the following index management methods, the table definition is marked as modified. For the changes to be reflected on the data and index files, you must call `CTTable::Alter()` method.

To add or delete an index from a table, call `CTTable::AddIndex()` and `CTTable::DelIndex()`. To edit index properties call `CTIndex::SetEmptyChar()`, `CTIndex::SetDuplicateFlag()`, and `CTIndex::SetNullFlag()`.

To add, insert or delete index segments from an index, call one of the overloaded methods `CTTable::AddSegment()`, `CTTable::InsSegment()`, or `CTTable::DelSegment()`.

Most changes relating to indices will trigger the `CTTable::Alter()` method to perform only an index rebuild. If only one index if affected, `CTTable::Alter()` will only rebuild the affected index. If changes affect more than one index, `CTTable::Alter()` may rebuild all indices.

After a successful alter table, all records associated with the altered table will automatically re-initialize to reflect any new table field and index definitions.

**Alter the table**

`CTTable::Alter()` scans the table, field, index, and segment structures to decide which changes need to be made and how to do it. At the very least, it may only update the table DODA if the only change done was, for example, in field types that are compatible with each other: changing from types CT_INT4 and CT_INT4U. Then, if the only changes occurred in a single index: a single index was added or deleted or the index properties changed, only that index is rebuilt. If more than one index changed, or more than one index was added or deleted, then it may be necessary to rebuild all indices of the table. If fields were added, deleted, inserted, or the changes in the field property were not compatible with each other, then Alter needs to perform a full rebuild of the table.

A table is rebuilt by creating a temporary table with the correct current properties taking in consideration all changes. All records are read from the original table and written into the temporary table. Once all data records have been moved, the original table is deleted and the temporary table is renamed with the name of the original table.

```cpp
// add one field to the table and rebuild it
CTTable ATable(ADatabase);

// open the table
ATable.Open("MyTable", CTOPEN_NORMAL);

// add one field to the table
ATable.AddField("Wages", CT_CURRENCY, 8);

// alter the table
try
    {
        ATable.Alter(CTDB_ALTER_NORMAL);
```
catch (CTException &err)
{
    printf("Alter table failed with error %d\n", err.GetErrorCode());
}

CTTable::Alter() method take as parameter an alter table action parameter:

<table>
<thead>
<tr>
<th>Action</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTDB_ALTER_NORMAL</td>
<td>0</td>
<td>Check for table changes before altering the table and perform only the changes required.</td>
</tr>
<tr>
<td>CTDB_ALTER_INDEX</td>
<td>1</td>
<td>Force rebuild of all indexes, regardless of table changes.</td>
</tr>
<tr>
<td>CTDB_ALTER_FULL</td>
<td>3</td>
<td>Force full table rebuild, regardless of table changes.</td>
</tr>
<tr>
<td>CTDB_ALTER_PURGEDUP</td>
<td>4096</td>
<td>Purge duplicate records</td>
</tr>
<tr>
<td>CTDB_ALTER_TRUNCATE</td>
<td>8192</td>
<td>Quickly remove all records</td>
</tr>
</tbody>
</table>

**Default Values**

c-treeDB’s alter table function can be used to alter the schema of an existing table by adding new fields or modifying existing fields of the specified table. During an alter table operation, when a new field is added to the table, or when an existing field type is changed, an optional default field value can be specified for these fields.

The default value of a field is used during an alter table operation when a full table rebuild is performed. During a full alter table rebuild, and after the old record buffer data is moved to the new record buffer, the new record buffer is scanned and, if a NULL field is found and that NULL field has a default value, the default value is copied to the field buffer. Typically the default field value is applied for new fields added to the table and to existing fields that have their types changed and the field value is NULL.

The field default value is kept as a string representation of the data. It is recommended that numeric data should be converted to string using one of the rich set of c-treeDB data conversion functions. Binary data can also be used by passing the pointer to data and the appropriate length.

The default value is set by calling the **CTField::SetFieldDefaultValue()** method.

**Example:**

```c
// set the default value of country field */
try
{
    CTFIELD hField = hTable.GetField("country");
    hField.SetFieldDefaultValue("USA");
}
catch (CTException &err)
{
    printf("SetFieldDefaultValue failed\n");
}
```

Use **GetDefaultfieldValue()** to retrieve the current field default value.
Example

// check if default field value is 'USA'
try
{
    CTString value;
    CTFIELD hField = hTable.GetField("country");

    hField = ctdbGetField(hTable, 5);
    if (hField.GetFieldDefaultValue(value) > 0)
    {
        if (value == "USA")
            printf("Default value is USA\n");
        else
            printf("Default value is not USA\n");
    } else
        printf("No default value set\n");
}
catch (CTException &err)
{
    printf("GetFieldDefaultValue failed\n");
}

You can check if a default value is set by calling the IsFieldDefaultValueSet() method.

Example

// check if default field value is set
CTFIELD hField = hTable.GetField("country");
if (hField.IsFieldDefaultValueSet())
    printf("Default field value is set\n");
else
    printf("No default field value\n");

Once set, a default field value will remain in place until the table handle is closed. The ClearFieldDefaultValue() method clears the default value associated with a field. The default date and time types are also reset to their default values of CTDATE_MDCY and CTTIME_HMS respectively.

Example

// clear the default field value
try
{
    CTFIELD hField = hTable.GetField("country");
    hField.ClearFieldDefaultValue();
}
catch (CTException &err)
{
    printf("ClearFieldDefaultValue failed\n");
}

You can clear the default values for all fields in a table by calling the ClearAllFieldDefaultValue() method.
Example

```c++
// clear all default field values
try
{
    hTable.ClearAllFieldDefaultValue();
}
catch (CTException &err)
{
    printf("ClearAllFieldDefaultValue failed\n");
}
```

The default date and time types used for conversions to and from strings can be changed by calling the `SetFieldDefaultDateTimeType()` method.

When setting the default field values with date, time or timestamp data, the data must be first converted to string. By default the date type is `CTDATE_MDCY` while the default time type is `CTTIME_HMS`.

The possible date formats for string conversion are:

<table>
<thead>
<tr>
<th>c-treeDB Symbolic Constant</th>
<th>c-treeDB .NET Symbolic Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CTDATE_MDCY</code></td>
<td><code>MDCY_DATE</code></td>
<td>Date is mm/dd/ccyy</td>
</tr>
<tr>
<td><code>CTDATE_MDY</code></td>
<td><code>MDY_DATE</code></td>
<td>Date is mm/dd/yy</td>
</tr>
<tr>
<td><code>CTDATE_DMCY</code></td>
<td><code>DMCY_DATE</code></td>
<td>Date is dd/mm/ccyy</td>
</tr>
<tr>
<td><code>CTDATE_DMY</code></td>
<td><code>DMY_DATE</code></td>
<td>Date is dd/mm/yy</td>
</tr>
<tr>
<td><code>CTDATE_CYMD</code></td>
<td><code>CYMD_DATE</code></td>
<td>Date is ccyymmdd</td>
</tr>
<tr>
<td><code>CTDATE_YMD</code></td>
<td><code>YMD_DATE</code></td>
<td>Date is yymmdd</td>
</tr>
</tbody>
</table>

Time Types can be one of the following string time formats:

<table>
<thead>
<tr>
<th>c-treeDB Symbolic Constant</th>
<th>c-treeDB .NET Symbolic Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CTTIME_HMSP</code></td>
<td><code>HMSP_TIME</code></td>
<td>Time is hh:mm:ss am</td>
</tr>
<tr>
<td><code>CTTIME_HMP</code></td>
<td><code>HMP_TIME</code></td>
<td>Time is hh:mm am</td>
</tr>
<tr>
<td><code>CTTIME_HMS</code></td>
<td><code>HMS_TIME</code></td>
<td>Time is hh:mm:ss (24 hour)</td>
</tr>
<tr>
<td><code>CTTIME_HM</code></td>
<td><code>HM_TIME</code></td>
<td>Time is hh:mm (24 hour)</td>
</tr>
<tr>
<td><code>CTTIME_MIL</code></td>
<td><code>MIL_TIME</code></td>
<td>Time is hhmm (military)</td>
</tr>
<tr>
<td><code>CTTIME_HHMST</code></td>
<td></td>
<td>Time is hh:mm:ss.ttt (24 hour)</td>
</tr>
</tbody>
</table>

Example

```c++
// set the field default date and time types
try
{
    
```
CTField hField = hTable.GetField("country");
hField.SetFieldDefaultDateTimeType();
} catch (CTException &err)
{   printf("SetFieldDefaultDateTimeType failed\n");
}

The default date type value can be retrieved by calling the **GetFieldDefaultDateType()** method.

**Example**

// check the default date type
try
{   CTField hField = hTable.GetField("country");
    if (hField.GetFieldDefaultDateType() == CTDATE_MDCY)
        printf("Field default date type is OK\n");
} catch (CTException &err)
{   printf("GetFieldDefaultDateType() failed\n");
}

The default time type value can be retrieved by calling the **GetFieldDefaultTimeType()** method.

**Example**

// check the default date type
try
{   CTField hField = hTable.GetField("country");
    if (hField.GetFieldDefaultTimeType() == CTTIME_HMS)
        printf("Field default time type is OK\n");
} catch (CTException &err)
{   printf("GetFieldDefaultTimeType() failed\n");
}

**Adding an index to a table**

If you need to add one or more indices to an existing table, perform the following steps:

1. **Add the index by calling** `CTTable::AddIndex()`. Repeat this step for each new index.
2. **Add, insert, or delete index segments** by calling the overloaded methods `CTTable::AddSegment()`, `CTTable::InsertSegment()`, or `CTTable::DelSegment()`. Repeat this step for each segment of the index.
3. **Call** `CTTable::Alter()` **to add the new index**

// add new index to table
CTTable ATable(ADatabase);

// open the table
ATable.Open("MyTable", CTOPEN_NORMAL);

// add the new index
ATable.AddIndex("MyNewIndex", CTINDEX_FIXED, YES, NO);
// add new index segments
ATable.AddSegment("MyNewIndex", "Field1", CTSEG_SCHSEG);

// alter the table to commit index changes to disk
try
{
    ATable.Alter(CTDB_ALTER_NORMAL);
}
catch (CTExplorer &err)
{
    printf("Alter table failed with error %d\n", err.GetErrorCode());
}

### Deleting an index from a table

If you need to delete one or mode indices from a table, perform the following steps:

1. Delete the index by calling `CTTable::DelIndex()`. There is no need to delete the index segments. Repeat this step for each index you want to delete.

2. Call `CTTable::Alter()` to delete the index from the table.

```c
// delete the first index
CTTable ATable(ADatabase);

// open the table
ATable.Open("MyTable", CTOPEN_NORMAL);

// delete the first index - MyIndex0
ATable.DelIndex("MyIndex0");

// alter the table to commit index changes to disk
try
{
    ATable.Alter(CTDB_ALTER_NORMAL);
}
catch (CTExplorer &err)
{
    printf("Alter table failed with error %d\n", err.GetErrorCode());
}
```

### Forcing an index rebuild

There may be situations where you may need to build the indices of a table. You can use the `CTDB_ALTER_INDEX` action parameter of `CTTable::Alter()` method to force the rebuild of all indices of a table. When `CTDB_ALTER_INDEX` is specified, `CTTable::Alter()` will rebuild all indices of a table regardless of any changes done to the table specification.

```c
// rebuild all indices
CTTable ATable(ADatabase);

// open the table
ATable.Open("MyTable", CTOPEN_NORMAL);

// rebuild all indices
// alter the table to commit index changes to disk
try
{
    ATable.Alter(CTDB_ALTER_INDEX);
}
catch (CTExplorer &err)
{
    printf("Alter table failed with error %d\n", err.GetErrorCode());
}
```
Forcing a table rebuild

There may be situations where you may need to force a full table rebuild. Please remember that in a full table rebuild, a temporary table is created based on the properties of the original table, then all records are read from the original table and written into the temporary table. All indices are also rebuilt. Once all data records have been moved, the original table is deleted and the temporary table is renamed with the name of the original table.

// rebuild a table
CTTable ATable(ADatabase);

// open the table
ATable.Open("MyTable", CTOPEN_NORMAL);

// rebuild the table
try
{
    ATable.Alter(CTDB_ALTER_FULL);
}
catch (CTExplorer &err)
{
    printf("Alter table failed with error %d\n", err.GetErrorCode());
}

Attach and Detach Open Tables

A c-treeDB table handle or object can be attached and detached to an already open data file. Applications may need to open a table using c-tree ISAM and low level functions and then attach the table to a c-treeDB table handle to take advantage of full c-treeDB functionality.

CTTable::Attach() attaches a c-tree Plus ISAM datno object to a c-treeDB table handle. This function is useful if you have opened a data or index file using one of c-tree’s ISAM open functions and need to attach it to a table handle to use some of the advanced c-treeDB features such as alter table or the record handler.

AttachXtd() attaches a c-tree Plus ISAM datno object to a c-treeDB table handle. This function is useful if you have opened a data and index file using one of c-tree’s ISAM open functions and need to attach it to a table handle to use some of the advanced c-treeDB features such as alter table or the record handler. This extended version allows the user to specify the DODA and IFIL for the table, enabling tables without DODA and/or IFIL to be attached to c-treeDB.

Detach() detaches a c-treeDB table handle from a c-tree data and index files. The table is not closed but the c-treeDB table handle resources are released and the handle re-initialized.

Example
/* DODA */
static DATOBJ doda[] =
{
    {"f1", (pTEXT)0, CT_INT4, 4},
    {"f2", (pTEXT)4, CT_FSTRING, 10}
};

/* IFIL */
static ISEG iseg = {0, 4, 12};
static IIDX iidx = {4, 0, 0, 0, 1, &iseg, "i311x1", NULL, NULL, NULL};

```c
static IFIL ifil = {"test310", -1, 14, 0, 0, 1, 0, 0, &iidx, "f1", "f2", 0};

CTHANDLE hSession = ctdbAllocSession(CTSESSION_CTREE);
CTHANDLE hTable = ctdbAllocTable(hSession);
CTHANDLE hRecord = ctdbAllocRecord(hTable);
NINT datno, count = 0;

/* logon to c-tree */
ctdbLogon(hSession, SERVER, USER, PASSWD);

/* open the file using c-tree ISAM */
datno = (NINT)OPNRFILX((COUNT) -1, "test309.dat", (COUNT)0, NULL);

/* attach to table */
ctdbAttachTableXtd(hTable, datno, doda, &ifil);

/* read the records */
if (ctdbFirstRecord(hRecord) == CTDBRET_OK)
do
{
    count++;
}
while (ctdbNextRecord(hRecord) == CTDBRET_OK);

/* cleanup */
ctdbDetachTable(hTable);
ctdbFreeRecord(hRecord);
ctdbFreeTable(hTable);
ctdbLogout(hSession);
ctdbFreeSession(hSession);

CTDBRET ctdbDetachTable(CTHANDLE Handle);
void CTTable::Detach();
```

**Table Properties**

c-treeDB tables have a number of properties that can be set when the table is created or with "set" and "get" functions.

**Table name**

The table name is a read only property as it can’t be changed once the table is created. Use **CTTable::GetName()** to retrieve the table name.

**Table Path**

The table path property is by default set to NULL. If this property is not changed prior to a **CTTable::Create()** call, the table is created in the same directory the database is located. Change the table path property with **CTTable::SetPath()**. **CTTable::GetPath()** retrieves the current table path.
Table file extension
The table data file extension is by default set to ".dat". Change the default data file extension with `CTTable::SetDataExtension()`. `CTTable::GetDataExtension()` retrieves the current data file extension for the table.

Index file extension
The table index file extension is by default set to ".idx". Change the default index file extension with `CTTable::SetIndexExtension()`. `CTTable::GetIndexExtension()` retrieves the current index file extension for the table.

Data file extent size
The data file extent is the default size by which the data file is extended when necessary. This value is 0 bytes by default. If there is the need to change it, use `CTTable::SetDataDefaultExtentSize()`. To retrieve the index default extent size, use `CTTable::GetDataDefaultExtentSize()`.

Index file extent size
The index file extent is the default size by which the index file is extended when necessary. This value is 0 bytes by default. If there is the need to change it, use `CTTable::SetIndexDefaultExtentSize()`. To retrieve the index default extent size, use `CTTable::GetIndexDefaultExtentSize()`.

Table password
By default tables do not have a password, i.e. the password property is set to NULL. If a table is to be created with a password, change this property before creating the table. To set the password use `CTTable::SetPassword()`, and `CTTable::GetPassword()` to retrieve it.

Table group ID
The group ID can be used to manage table permissions for multiple users. By default, no group ID settings are specified for c-treeDB tables, i.e. the default group ID value is NULL. If a table is to be created with a group ID, set the value of this property before creating the table. To set a group ID, use `CTTable::SetGroupid()` and `CTTable::GetGroupid()` to retrieve it.
### Table permission mask

The permission mask is set at table creation and specifies a permission mask that determines the kind of access that users may acquire on subsequent opens. The mask is comprised of three components: owner permissions, group permissions and world permissions. With this structure, you are able to allow different users different levels of access to the file. The default permission mask is set to `(OPF_ALL | GPF_READ | GPF_WRITE | WPF_READ | WPF_WRITE)`. To set the table permissions, use `CTTable::SetPermission()`. To retrieve the permission mask, use `CTTable::GetPermission()`. The valid permission mask values are: c-treeDB Permission Constant

<table>
<thead>
<tr>
<th>c-treeDB .NET Permission Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPF_READ</td>
<td>O_READ  owner read permission</td>
</tr>
<tr>
<td>OPF_WRITE</td>
<td>O_WRITE  owner write/update permission</td>
</tr>
<tr>
<td>OPF_DEF</td>
<td>O_DEF  owner file definition permission</td>
</tr>
<tr>
<td>OPF_DELETE</td>
<td>O_DELETE  owner file deletion permission</td>
</tr>
<tr>
<td>OPF_ALL</td>
<td>O_ALL  owner granted all permissions</td>
</tr>
<tr>
<td>OPF_NOPASS</td>
<td>O_NOPASS  owner grants read only without password</td>
</tr>
<tr>
<td>GPF_NONE</td>
<td>G_NONE  group access denied</td>
</tr>
<tr>
<td>GPF_READ</td>
<td>G_READ  group read permission</td>
</tr>
<tr>
<td>GPF_WRITE</td>
<td>G_WRITE  group write/update permission</td>
</tr>
<tr>
<td>GPF_DEF</td>
<td>G_DEF  group file definition permission</td>
</tr>
<tr>
<td>GPF_DELETE</td>
<td>G_DELETE  group file deletion permission</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c-treeDB .NET Permission Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPF_READ</td>
<td>O_READ  owner read permission</td>
</tr>
<tr>
<td>OPF_WRITE</td>
<td>O_WRITE  owner write/update permission</td>
</tr>
<tr>
<td>OPF_DEF</td>
<td>O_DEF  owner file definition permission</td>
</tr>
<tr>
<td>OPF_DELETE</td>
<td>O_DELETE  owner file deletion permission</td>
</tr>
<tr>
<td>OPF_ALL</td>
<td>O_ALL  owner granted all permissions</td>
</tr>
<tr>
<td>OPF_NOPASS</td>
<td>O_NOPASS  owner grants read only without password</td>
</tr>
<tr>
<td>GPF_NONE</td>
<td>G_NONE  group access denied</td>
</tr>
<tr>
<td>GPF_READ</td>
<td>G_READ  group read permission</td>
</tr>
<tr>
<td>GPF_WRITE</td>
<td>G_WRITE  group write/update permission</td>
</tr>
<tr>
<td>GPF_DEF</td>
<td>G_DEF  group file definition permission</td>
</tr>
<tr>
<td>GPF_DELETE</td>
<td>G_DELETE  group file deletion permission</td>
</tr>
</tbody>
</table>
The permission mask is set at table creation and specifies a permission mask that determines the kind of access that users may acquire on subsequent opens. The mask is comprised of three components: owner permissions, group permissions and world permissions. With this structure, you are able to allow different users different levels of access to the file. The default permission mask is set to `(OPF_ALL | GPF_READ | GPF_WRITE | WPF_READ | WPF_WRITE)`. To set the table permissions, use `CTTable::SetPermission()`. To retrieve the permission mask, use `CTTable::GetPermission()`. The valid permission mask values are: c-treeDB .NET Permission Constant

<table>
<thead>
<tr>
<th>Permission Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPF_NOPASS</td>
<td>group read only access without password</td>
</tr>
<tr>
<td>WPF_NONE</td>
<td>world access denied</td>
</tr>
<tr>
<td>WPF_READ</td>
<td>world read permission</td>
</tr>
<tr>
<td>WPF_WRITE</td>
<td>world write/update permission</td>
</tr>
<tr>
<td>WPF_DEF</td>
<td>world file definition permission</td>
</tr>
<tr>
<td>WPF_DELETE</td>
<td>world file deletion permission</td>
</tr>
<tr>
<td>WPF_NOPASS</td>
<td>world read only access without password</td>
</tr>
</tbody>
</table>

Number of fields

The "number of fields" property indicates the total number of fields defined for a table. The value takes in consideration only user defined fields and does not include any hidden fields that exist for
the table. Use `CTTable::GetFieldCount()` method to retrieve the number of fields associated with a table.

**Number of indices**

The number of indices property indicates the total number of indices defined for a table. The value takes in consideration only user defined indices and does not include the `RECBYT` or `ROWID` indices. Use `CTTable::GetIndexCount()` method to retrieve the number of indices associated with a table.

**Field padding**

By default the c-treeDB API pad fixed length string fields `CT_CHARS (CT_FSTRING)`, `CT_FPSTRING`, `CT_F2STRING` and `CT_F4STRING` with Nulls (`\0`) bytes. The field padding property sets the table pad and field delimiter characters to allow proper target key formation. This property allows the c-treeDB API to operate on files created using the c-tree Plus ISAM and low level APIs using a fixed string padding strategy that is different from the c-treeDB API default.

Use `CTTable::SetPadChar()` to set the pad and field delimiter character.

`CTTable::GetPadChar()` retrieves the current pad and field delimiter character.

```cpp
// set the table pad and delimiter characters to spaces
try
{   
    ATable.SetPadChar(' ', ' ');
}
catch (CTException &err)
{
    printf("Set pad character failed with error %d\n", err.GetErrorCode());
}
```

The most common strategies for padding fixed string fields are:

- **Pad with Nulls**: pad character is `\0` and field delimiter is `\0`
- **Pad with spaces**: pad character is `' '` and field delimiter is `' '`
- **Pad with spaces and terminate with NUL**: pad character is `' '` and field delimiter is `\0`

**Update create mode**

Use the update table create mode property to change the table create mode after the table has been created. Use the function `ctdbUpdateCreateMode()` to change the table create mode. You can only update the create mode if the table was opened in exclusive mode.

```cpp
// create a new table object
CTTable ATable(ADatabase);

// open the table exclusive
ATable.Open("MyTable", CTOPEN_EXCLUSIVE);

// update the table create mode
try
{
    // retrieve the current create mode
    CTCREATE_MODE curmode = ATable.CreateMode();

    // add transaction processing
    curmode |= CTCREATE_TRNLOG;
}
```
// update the table create mode
ATable.UpdateCreateMode(CTCREATE_TRNLOG);
}
catch (CTException &err)
{
    printf("Update create mode failed with error %d\n", err.GetErrorCode());
}

CTTable::UpdateCreateMode() changes critical file mode attributes such as the level of transaction control. No check is made to determine if the mode change will damage data. No check is made if the new mode is valid. Use this function with caution as data may be lost. For instance, changing a data file from transaction processing to no transaction processing makes automatic recovery unavailable.

The mode parameter passed to CTTable::UpdateCreateMode() represents the new table create mode. It must be perfectly formed, as it will replace the current table create mode. Use the function ctdbGetTableCreateMode() to retrieve the current create mode and apply the changes on a fully qualified create mode. Update only the following create table modes:

- **CTCREATE_PREIMG**
- **CTCREATE_TRNLOG**
- **CTCREATE_WRITETHRU**
- **CTCREATE_CHECKLOCK**
- **CTCREATE_CHECKREAD**
- **CTCREATE_HUGEFILE**
3.5 Working with Records

The c-treeDB API record management hides from the user all the complexities of maintaining a generic record buffer for tables. While fixed length records may be easier to implement and maintain, variable length records do present a greater challenge for the developer.

When a table contains variable length records, each record read may require buffers of different sizes. The c-treeDB record manager performs the function of expanding and shrinking record buffers to deal with the necessities of variable length records. The c-tree ISAM API also requires different API calls for fixed length and variable length records, but the c-treeDB presents to the user one common API that deals with both fixed and variable length records.

Only the application architecture and system resources limit the number of c-treeDB record buffers associated with a table.

Creating a record object

A valid record object is required before most record operations can take place. You need to pass a valid CTTable object to the CTRecord constructor.

```c
// create a CTRecord object
CTRecord ARecord(ATable);
```

If you create a dynamic CTRecord object with the new operator, you are required to destroy the object with the delete operator.

```c
// create a dynamic CTRecord object
CTRecord* pRecord = new CTRecord(ATable);

if (!pRecord)
{
    printf("CTRecord creation failed\n");
}
... other operations ..
// destroy the CTRecord object
delete pRecord;
```

After the record object has been created, but before any record operations can take place, the table associated with the record object must be opened.

Record buffer layout

Based on the fields added by the user, c-treeDB will organize the record buffer as follows:

|$DELFLD$| $NULFLD$| $ROWID$| field 0 | field 1 | ... | field n |

The user fields are stored in the table record in the same order in which they were added. To initiate the operation on the records, a record handle must be declared and instantiated. Once the table associated with the record handle is opened, the record can be used to store the information that come from or go to the fields in the table.
Sharing the same context

Every time one of the `CTRecord` class constructors creates a record object, the record object acquires its own context, so that each record buffer operates independently. Record operations that move the current record position will not interfere with each other.

There are situations where it may be necessary to have several different record objects sharing the same record context. In this case the developer can create the first record buffer using one of the `CTRecord` class constructors to acquire a new context. The initial record can then be duplicated by creating a new record object and passing the original record object instead of a table object. The duplicated records will share the same context of the original record buffer, but the duplicated record handle will not share the same memory buffer for holding the record data.

```c
// create two records sharing the same context
CTRecord ARecord1(ATable);

// create the duplicated record object
CTRecord ARecord2(ARecord1);
```

In the code fragment above, the record object `ARecord1` was created passing a table object as the parent object. `ARecord1` is created with a new record context. `ARecord2` was created passing the `ARecord1` object as the parent object. `ARecord2` is a duplicated copy of `ARecord1`.

Resetting a record

A record object may be reset to its initial state, i.e. to the same state it had just after being created. The record reset function will perform the following actions on a record handle:

- The context associated with the record is released
- Record sets are turned off
- Memory allocated for record buffer is released
- Record properties are reinitialized.

```c
// reset a record
try
{
    ARecord.Reset();
}
catch (CTException &err)
{
    printf("Record reset failed with error %d\n", err.GetErrorCode());
}
```

All records associated with a table can be reset in one operation by calling `CTTable::ResetAll()` method.

```c
// reset all records
try
{
    ATable.ResetAll();
}
catch (CTException &err)
{
    printf("Reset all records failed with error %d\n", err.GetErrorCode());
}
```

When a table is closed, all records associated with that table are automatically reset by the `CTTable::Close()` method.
The default index

Some of the record operations such as record navigation, finding records and record sets require an index to be specified. The default index property is used by the record functions that required an index to operate on.

Any of the user defined indices, or `RECBYT`, or `ROWID`, in the case these are created, may be used as the index controlling the search. To set the default index use one of the overloaded `CTRecord::SetDefaultIndex()` methods. You can use the index number of the index name to identify the default index.

```c++
// create a new record object
CTRecord ARecord( ATable);
// set the default index to "MyIndex"
try {
   ARecord.SetDefaultIndex( "MyIndex" );
} catch (CTException &err) {
   printf("Set default index failed with error %d\n", err.GetErrorCode());
}
```

Use `CTRecord::GetDefaultIndex()` to retrieve the current index number or `CTRecord::GetDefaultIndexName()` to retrieve the name of the current default index.

```c++
// make sure the default index is set to "MyIndex"
try {
   if (ARecord.GetDefaultIndexName() != "MyIndex") {
      ARecord.SetDefaultIndex( "MyIndex" );
   }
} catch (CTException &err) {
   printf("Set default index failed with error %d\n", err.GetErrorCode());
}
```

If no index is set as the default index before the first search, the default index is the first user-defined index. Once an index is chosen as the default index, all subsequent searches will be done using this index, up to a new selection in the default index. For instance: the user may set the default index as the `ROWID` index, search for the first input record, then change the default index to the user defined index (i_name, for instance, an index over the name field in the table), and search for the next record on this particular sequence, taking as the previous record the first input record.

A user may perform a chronological search in the records in a table using `CTRecord::First()` and `CTRecord::Next()`, and setting the `ROWID` index as the default index:

Selecting the RECBYT index

The `RECBYT` index can be selected as the default index in order to perform record navigation with `CTRecord::First()`, `CTRecord::Last()`, `CTRecord::Next()` and `CTRecord::Prev()`, ordered by the record offset. The `RECBYT` index can be selected as the default index by passing the index value of `CTDB_RECBYTIDXNO` to `CTRecord::SetDefaultIndex()` method.
// set the RECBYT index as the default index
try
{
    ARecord.SetDefaultIndex(CTDB_RECBYT_IDXNO);
}
catch (CTException &err)
{
    printf("Set default index failed with error %d\n", err.GetErrorCode());
}

Selecting the ROWID index

The ROWID index can be selected as the default index in order to perform record navigation with
CTRecord::First(), CTRecord::Last(), CTRecord::Next() and CTRecord::Prev(), ordered by
the ROWID value of each record. The ROWID index can be selected as the default index by
passing the index value of CTDB_ROWID_IDXNO to CTRecord::SetDefaultIndex() method.

// set the ROWID index as the default index
try
{
    ARecord.SetDefaultIndex(CTDB_ROWID_IDXNO);
}
catch (CTException &err)
{
    printf("Set default index failed with error %d\n", err.GetErrorCode());
}

Navigating records

c-treeDB provides features that allow navigation among records of a particular table. You can
position the current record at the first record, last record, next record, previous record, or seek to
a specific record if you know the record offset.

Setting a different default index for the record may change the order of the record navigation.

The record navigation function not only updates the current position of a record, but the record
data is read from disk and placed in the record buffer inside the record handle. As the record is
read from disk, the new record flag is set to false to indicate that the data in the record buffer
originated from an existing record in the table.

First record

Call CTRecord::First() to position the current record at the first record of a table. If the table is
empty, i.e. the table contains no records, CTRecord::First() returns NO (false).

// get the first record of the table
try
{
    if (ARecord.First())
        printf("First record found\n");
    else
        printf("Table is empty\n");
}
catch (CTException &err)
{
    printf("First record failed with error %d\n", err.GetErrorCode());
}
Last record

Call `CTRecord::Last()` to position the current record at the last record of a table. If the table is empty, i.e. the table contains no records, `CTRecord::Last()` returns NO (false).

```c
// get the last record of the table
try
{
  if (ARecord.Last())
    printf("Last record found\n");
  else
    printf("Table is empty\n");
}
catch (CTException &err)
{
  printf("Last record failed with error %d\n", err.GetErrorCode());
}
```

Next record

Call `CTRecord::Next()` to position the current record at the next record of a table. If the current record is already the last record of a table, `CTRecord::Next()` returns NO (false).

```c
// get the next record of the table
try
{
  if (ARecord.Next())
    printf("Next record found\n");
  else
    printf("No more records\n");
}
catch (CTException &err)
{
  printf("Next record failed with error %d\n", err.GetErrorCode());
}
```

A current record must exist before `CTRecord::Next()` is called or an ICUR_ERR (100) is thrown indicating that there can be no next record is the current record does not exist. You can establish a current record by calling `CTRecord::First()`, `CTRecord::Last()`, `CTRecord::Find()` or `CTRecord::SeekRecord()`.

Previous record

Call `CTRecord::Prev()` to position the current record at the previous record of a table. If the current record is already the first record of a table, `CTRecord::Prev()` returns NO (false).

```c
// get the previous record of the table
try
{
  if (ARecord.Prev())
    printf("Previous record found\n");
  else
    printf("Already at first record\n");
}
catch (CTException &err)
{
  printf("Prev record failed with error %d\n", err.GetErrorCode());
}
```

A current record must exist before `CTRecord::Prev()` is called, or an ICUR_ERR (100) exception is thrown indicating that there can be no previous record if the current record does not exist. You
can establish a current record by calling `CTRecord::First()`, `CTRecord::Last()`, `CTRecord::Find()` or `CTRecord::SeekRecord()`.

**Seek to record**

If you know in advance the offset of a given record, **`CTRecord::SeekRecord()`** can be used to make it the current record.

You can use **`CTRecord::GetRecordPos()`** and **`CTRecord::SeekRecord()`** to implement a bookmark system for your records. Retrieve the record offset with **`CTRecord::GetRecordPos()`**, which is the bookmark value, and later on you can quickly go back to the record by calling **`CTRecord::SeekRecord()`**.

```cpp
// get record bookmark
CTOFFSET GetBookmark(CTRecord &ARecord)
{
    return ARecord.GetRecordPos();
}

// goto record bookmark
void GotoBookmark(CTRecord &ARecord, CTOFFSET bookmark)
{
    ARecord.SeekRecord(bookmark);
}
```

**Finding records**

You can search for records in a table using one of the **`CTRecord::Find()`**, **`CTRecord::FindTarget()`** and **`CTRecord::FindRowid()`** methods. c-treeDB performs the find operations against the table indices. When an index entry is found, the record data is loaded into the record buffer.

Before you can call **`CTRecord::Find()`**, you need to prepare the data you want to find:

1. Clear a record buffer
2. Set the default index with the appropriate value
3. Populate the fields that makeup the index segment
4. Call **`CTRecord::Find()`** with the appropriate find mode

```cpp
// find record which product code is DISKETTE
CTRecord ARecord(ATable);

// clear the record buffer
ARecord.Clear();

// set the default index to "MyIndex"
ARecord.SetDefaultIndex("MyIndex");

// populate the 'product' field
ARecord.SetFieldAsString("product", "DISKETTE");

// find the record
try
{
    if (ARecord.Find(CTFIND_EQ))
    {
        printf("Record found\n");
    }
    else
    {
        printf("Record not found\n");
    }
```

catch (CTException &err)
{
    printf("Record find failed with error %d\n", err.GetErrorCode());
}

CTRecord::Find(), CTRecord::FindTarget() and CTRecord::FindRowid() returns YES (true) if the record was found or NO (false) if the record was not found. These methods throw an exception if an error occurs.

**Find Modes**

Use the following find modes with the record find methods:

<table>
<thead>
<tr>
<th>c-treeDB Find mode</th>
<th>c-treeDB .NET Find Mode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTFIND_EQ</td>
<td>EQ</td>
<td>Find a record equal to the target</td>
</tr>
<tr>
<td>CTFIND_LT</td>
<td>LT</td>
<td>Find a record less than target</td>
</tr>
<tr>
<td>CTFIND_LE</td>
<td>LE</td>
<td>Find a record less or equal than target</td>
</tr>
<tr>
<td>CTFIND_GT</td>
<td>GT</td>
<td>Find a record greater than target</td>
</tr>
<tr>
<td>CTFIND_GE</td>
<td>GE</td>
<td>Find a record greater or equal than target</td>
</tr>
</tbody>
</table>

**Note:** The Find Mode CTFIND_EQ requires that the target contains values for all segments that compose the index and the index cannot allow duplicates.

**Note:** c-treeDB .NET defines this mode with the FIND_MODE enum.

### Building a target key

A target key can be built using most of the steps used to find a record using CTRecord::FindTarget() method. To build a target key you must:

1. Clear a record buffer
2. Set the default index with the appropriate value
3. Populate the fields that makeup the index segment
4. Call CTRecord::BuildTargetKey() with the appropriate find mode and the buffer to receive the target key

```c
// build a target key
CTRecord ARecord(ATable);
TEXT key[256];
VRLEN keylen = sizeof(key);

// clear the record buffer
ARecord.Clear();

// set the default index to index 0
ARecord.SetDefaultIndex(0);

// populate the 'product' field
ARecord.SetFieldAsString("product", "DISKETTE");

// build the target key
ARecord.BuildTargetKey(CTFIND_EQ, key, keylen);
```
try
{
    ARecord.BuildTargetKey(CTFIND_EQ, key, &keylen);
}
catch (CTException &err)
{
    printf("Build target key failed with error %d\n", err.GetErrorCode());
}

Finding records by target key

A record can also be found by passing a target key already built by calling
`CTRecord::FindTarget()` method. `CTRecord::FindTarget()` takes as its parameters a pointer to
a target key and the find mode.

```c
// find record with a target key
CTRecord ARecord(ATable);
TEXT key[256];
VRLEN keylen = sizeof(key);

// clear the record buffer
ARecord.Clear();

// set the default index to index 0
ARecord.SetDefaultIndex(0);

// populate the 'product' field
ARecord.SetFieldAsString("product", "DISKETTE");

// build the target key
ARecord.BuildTargetKey(CTFIND_EQ, key, &keylen);
try
{
    if (ARecord.FindTarget(key, CTFIND_EQ))
        printf("Record found\n");
    else
        printf("Record not found\n");
}
catch (CTException &err)
{
    printf("Find target failed with error %d\n", err.GetErrorCode());
}
```

Finding records by ROWID

A record can be quickly located if the `ROWID` for that record is known by calling
`CTRecord::FindRowid()`. `CTRecord::FindRowid()` take as its parameters the `ROWID` value and
the find mode.

```c
// find record with rowid of 1000
CTRecord ARecord(ATable);
try
{
    if (ARecord.FindRowid(1000, CTFIND_EQ))
        printf("Record found\n");
    else
        printf("No record found\n");
}
catch (CTException &err)
{
    printf("Find Rowid failed with error %d\n", err.GetErrorCode());
}
Record sets

An alternative and more efficient way to retrieve records is the use of Record Sets. A record set contains part of a table, representing a group of records that have keys that match a certain target. For example, an index contains the following keys:

CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK CASE DISK

When scanning through the index looking for records, record navigation functions work on the entire index file. Navigating to the first record obtains the first key in the index, CASE, and moving to the next record obtains each of the following keys in turn, on through KEYCAP.

When creating a record set using `CTRecord::RecordSetOn()`, with a target of "DISK" and a target length of 4, c-treeDB returns only the records in which the first 4 bytes of the key start with "DISK".

The record navigation functions will operate only on the records that match the set until the record set operation is terminated with a call to `CTRecord::RecordSetOff()`. To work with multiple record sets at once, use more than one record handle.

Record sets allow a faster search for records, particularly in client/server mode since fewer records may be retrieved and less traffic on the network will be generated. When used in conjunction with filters (see below), one can create simple queries.

To use record sets, follow the steps below:

1. Clear the record buffer
2. Set the index you want to use
3. Populate the index segments you would like to participate in the set
4. Establish the Set

Creating a record set

To create a record set, perform the following steps

1. Clear a record handle
2. Select the default index
3. Populate the field, or fields, that form the partial index
4. Call `CTRecord::RecordSetOn()` to create a record set

```c++
// create a new record object
CTRecord ARecord(ATable);
// clear the record buffer
ARecord.Clear();
// populate the fields
ARecord.SetString(0, "DISK");
// create the record set
try
{
    ARecord.RecordSetOn(4);
}
catch (CTException &err)
{
    printf("Record set on failed with error %d\n", err.GetErrorCode());
}
```
Once the record set is created, `CTRecord::First()`, `CTRecord::Next()`, `CTRecord::Last()` and `CTRecord::Prev()` will return only the records that match the record set criteria.

**Terminating a record set**

Once the record set is no longer necessary, you can turn it off by calling `CTRecord::RecordSetOff()`.

```cpp
// terminate the record set
try {
    ARecord.RecordSetOff();
} catch (CTException &err) {
    printf("Record set off failed with error %d\n", err.GetErrorCode());
}
```

Once the record set is terminated, `CTRecord::First()`, `CTRecord::Next()`, `CTRecord::Last()` and `CTRecord::Prev()` will return all records of a table.

**Record Filters**

c-treeDB allows users to define record filters using `CTTable::FilterRecord()`. When a filter is set, all records retrieved from a table are filtered against the given expression, and only records matching filter criteria are returned.

```cpp
// set filter
try {
    ATable.FilterRecord("ZIPCODE == 12345");
} catch (CTException &err) {
    printf("Filter record failed with error %d\n", err.GetErrorCode());
}
```

Only the user who sets the filter will have its records filtered. The filter is turned off when the table is closed, or when `CTTable::FilterRecord()` is called with an empty string as the filter expression. Only one filter may be active per table per user at once, so if a new filter is set to a table with an existing filter for the specific table, the old filter will be overridden.

```cpp
// terminate filter
try {
    CTString empty;
    ATable.FilterRecord(empty);
} catch (CTException &err) {
    printf("Filter record failed with error %d\n", err.GetErrorCode());
}
```

Use `CTTable::GetFilter()` to retrieve the current filter expression. If no filters are active, `CTTable::GetFilter()` return an empty string. Use `CTTable::IsFilteredRecord()` to test if filters are active for a table.
When used in the client/server model, this feature has the potential to increase the performance since just the records matching the criteria will be returned, reducing the network traffic.

Record filters, when used in conjunction with sets provide the c-treeDB API with simple query capabilities.

**c-treeACE Expression Parser and Grammar**

A powerful expression parser/analyzer provides for complex conditional expressions that can be defined and evaluated at run time.

Filter expression syntax closely follows the C language syntax for expressions, including order of precedence. An expression interpreted by the expression parser should compile without errors with a standard C compiler. As in C, you cannot compare strings directly like LastName > 'S'. However, the expression parser has a number of built-in functions that allow the comparison of strings. Example:

```
strcmp( LastName, "S" ) > 0
```

The expression handling assures proper alignment considerations are handled, and ensures buffer size of any record being evaluated is big enough.

Routines that evaluate conditional expressions maintain fixed data record lengths and total data record lengths. This permits correct alignment adjustments and detects if insufficient data is available. The latter condition results in a **CVAL_ERR** (598) error. The easiest way to produce a **CVAL_ERR** (598) is to read only the fixed-length portion of a data record, and have an expression that relies on fields in the variable-length portion of the record.

For additional control, a **Conditional Expression Callback Function** (page 142) is available. This allows advanced control through an external user created function.

**See also**

- *Data Filters* ([https://docs.faircom.com/doc/ctreeplus/30429.htm](https://docs.faircom.com/doc/ctreeplus/30429.htm))
- *Conditional Index Support*
- *Partitioned Files*
- *cndxparse*
- *cndxeval*
- *cndxrun*
- *cndxfree*
- *ctparsedoda*
- *getcndxmem*
- *putcndxmem*

**Constants**

The expression parser uses the constants below internally as a 32-bit signed or unsigned integer:

- **Character constants**: any valid char enclosed in single quotes, e.g., ‘a’
- **Signed Integer constants**: values from -2,147,438,647 to 2,147,438,647
- **Unsigned Integer constants**: values from 0 to 4,294,967,295
Hexadecimal constants: values from 0x00000000 to 0xffffffff. Any combination of lower case or upper case letters are accepted, e.g., 0Xbc4f or 0xF5C56d

Any integer larger than the maximum size allowed for integers, or any number with decimal points, or any numbers in scientific notation are interpreted as a floating point constant by the expression parser.

String constants are similar to C string constants and represent any text enclosed by double quotes, for example, ”This is a string”. The maximum size of a string constant defaults to 255 characters. The filter expression parser allows the following escape characters in string constants or character constants:

<table>
<thead>
<tr>
<th>Escape char</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\a</code> or <code>\A</code></td>
<td>ASCII 7</td>
<td>bell</td>
</tr>
<tr>
<td><code>\b</code> or <code>\B</code></td>
<td>ASCII 8</td>
<td>backspace</td>
</tr>
<tr>
<td><code>\f</code> or <code>\F</code></td>
<td>ASCII 12</td>
<td>Form Feed</td>
</tr>
<tr>
<td><code>\n</code> or <code>\N</code></td>
<td>ASCII 10</td>
<td>Linefeed</td>
</tr>
<tr>
<td><code>\r</code> or <code>\R</code></td>
<td>ASCII 13</td>
<td>Carriage Return</td>
</tr>
<tr>
<td><code>\t</code> or <code>\T</code></td>
<td>ASCII 9</td>
<td>tab</td>
</tr>
<tr>
<td><code>\v</code> or <code>\V</code></td>
<td>ASCII 11</td>
<td>vertical tab</td>
</tr>
<tr>
<td><code>\</code></td>
<td>Any</td>
<td>Any character not listed above</td>
</tr>
</tbody>
</table>

Variables

A filter expression variable is actually the name of the fields defined for the table. There is a limit of 128 characters for the name of variables and the names are case sensitive.

When a user specifies a variable name, the filter parser searches the table definition for a field of that name. The parser uses the type of the field and converts it to the types used internally by the expression evaluator. The conversion of field types is as follows:

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Data Type</th>
<th>Field Type</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT_BOOL</td>
<td>int</td>
<td>CT_SFLOAT</td>
<td>double</td>
</tr>
<tr>
<td>CT_CHAR</td>
<td>int</td>
<td>CT_DFLOAT</td>
<td>double</td>
</tr>
<tr>
<td>CT_CHARU</td>
<td>unsigned</td>
<td>CT_FSTRING</td>
<td>char*</td>
</tr>
<tr>
<td>CT_INT2</td>
<td>int</td>
<td>CT_FPSTRING</td>
<td>char*</td>
</tr>
<tr>
<td>CT_INT2U</td>
<td>unsigned</td>
<td>CT_F2STRING</td>
<td>char*</td>
</tr>
<tr>
<td>CT_INT4</td>
<td>int</td>
<td>CT_F4STRING</td>
<td>char*</td>
</tr>
<tr>
<td>CT_INT4U</td>
<td>unsigned</td>
<td>CT_STRING</td>
<td>char*</td>
</tr>
</tbody>
</table>
Field names that match a valid expression reserved word:
Consider a field named "year", which collides with the function YEAR. The expression "[year] == 2000" is needed to handle "year" as a field name rather than the function.

Parentheses
Use parentheses exactly like they are used in C expressions. There are no limits on the number of parentheses you may use in an expression, as long as each open parenthesis has a closing parenthesis. Parentheses are also used to enclose the arguments of built-in functions.

Predefined Functions
The c-treeACE conditional expression parser has numerous built-in functions for advanced conditional filtering possibilities. These are arranged and described in the following categories:

- C Language Equivalents (page 125)
- String and Text Functions (page 126)
- Date and Time Functions (page 131)
- Mathematical Functions (page 133)

C Language Equivalents
These built-in functions closely follow C library functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>int atoi( char* String )</td>
<td>convert string to integer</td>
</tr>
<tr>
<td>int atol( char* String )</td>
<td>convert string to long</td>
</tr>
<tr>
<td>double atof( char* String )</td>
<td>convert string to double</td>
</tr>
<tr>
<td>int cabs( into Value )</td>
<td>return the absolute value of a complex number.</td>
</tr>
<tr>
<td>int labs( into Value )</td>
<td>return the absolute value of a long integer.</td>
</tr>
<tr>
<td>double fabs( double Value )</td>
<td>return the absolute value of a double</td>
</tr>
<tr>
<td>double ceil( double Value )</td>
<td>return the ceiling of a value</td>
</tr>
<tr>
<td>double floor( double Value )</td>
<td>return the floor of a value</td>
</tr>
<tr>
<td>double fmod( double r, double t )</td>
<td>return the remainder of r by t as a double</td>
</tr>
</tbody>
</table>
### Function Explanation

<table>
<thead>
<tr>
<th>Function</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int strlen(char* String)</code></td>
<td>return the length of a string</td>
</tr>
<tr>
<td><code>int strcmp(char* s, char* t)</code></td>
<td>compare two strings</td>
</tr>
<tr>
<td><code>int stricmp(char* s, char* t)</code></td>
<td>compare strings without regard to case</td>
</tr>
<tr>
<td><code>int strncmp(char* s, char* t, int len)</code></td>
<td>compare characters of two strings</td>
</tr>
<tr>
<td><code>int strnicmp(char* s, char* t, int len)</code></td>
<td>compare characters of two strings without regard to case</td>
</tr>
<tr>
<td><code>int memcmp(char* s, char* t, int len)</code></td>
<td>compare bytes of two memory locations</td>
</tr>
<tr>
<td><code>TIME62CTDATE</code></td>
<td>convert integer time_t value to a c-tree Date type (CTDATE)</td>
</tr>
<tr>
<td><code>TIME2CTTIME</code></td>
<td>convert integer time_t value to a c-tree Time type (CTTIME)</td>
</tr>
<tr>
<td><code>TIME2CTTIMES</code></td>
<td>convert integer time_t value to a c-tree Timestamp type (CTTIMES)</td>
</tr>
</tbody>
</table>

### String and Text Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ASCII (char_expression)</code></td>
<td><code>int</code></td>
<td>Returns the ASCII value of the first character of <code>char_expression</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>char_expression</code> must be a character type value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <code>char_expression</code> is null, result is NULL.</td>
</tr>
<tr>
<td><code>CHR (integer_expression)</code></td>
<td><code>text</code></td>
<td>Returns a character string with the first character having an ASCII value equal to the argument expression.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>integer_expression</code> must evaluate to INTEGER, SMALLINT or TINYINT data type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <code>integer_expression</code> is NULL <code>CHR</code> returns NULL.</td>
</tr>
<tr>
<td><code>CONCAT (char_expression1, char_expression2)</code></td>
<td><code>text</code></td>
<td>Returns a concatenated character string formed by concatenating <code>char_expression1</code> with <code>char_expression2</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Both arguments must evaluate to Character type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If either expression is null, result is NULL.</td>
</tr>
<tr>
<td><code>FIELD (offset, size)</code></td>
<td><code>text</code></td>
<td>Returns the content of record at <code>offset</code> with <code>length</code> size as if it were a CT_ARRAY field.</td>
</tr>
<tr>
<td>Function</td>
<td>Return Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| `INSERT ( char_expression1, start, length, char_expression2 )` | text        | Returns a character string where `length` characters have been deleted from `char_expr1` beginning at `start` and `char_expr2` has been inserted into `char_expr1`, beginning at `start`.  
• `char_expr1` and `char_expr2` must evaluate to a character value  
• `start` and `length` must evaluate to INTEGER values  
• If either expression is null result is NULL. |
| `LEFT ( char_expression, count )`             | text        | Returns the leftmost `count` of characters of `char_expression`.  
• `char_expression` must evaluate to a CHAR value.  
• `count` must evaluate to an INTEGER value.  
• If `count` is less than one, an empty string is returned.  
• If `count` is greater than the length of `char_expression`, `char_expression` is returned.  
• If `char_expression` is null, LEFT returns NULL. |
| `LENGTH ( char_expression )`                  | int         | Returns the number of characters in `char_expression`.  
• `char_expression` must evaluate to a CHAR value.  
• If `char_expression` is null LENGTH returns NULL. |
| `LOCATE ( char Expr1, char Expr2 [ , start ] )` | int         | Returns the location of the first occurrence of `char_expr1` in `char_expr2`. If the function includes an optional integer argument `start` LOCATE begins searching `char_expr2` at that position. If the function omits `start`, LOCATE begins its search at the beginning of `char_expr2`. LOCATE denotes the first character position of a character expression as 1. If the search fails, LOCATE returns 0.  
• `char_expr1` and `char_expr2` must evaluate to a CHAR value.  
• `start`, if supplied, must evaluate to an INTEGER value.  
• If either expression is null, LOCATE returns NULL. |
| `LOWER ( char_expression )`                   | text        | Returns the result of the character expression after converting all characters to lowercase.  
• `char_expression` must evaluate to a CHAR value.  
• If `char_expression` is null LOWER returns NULL. |
<table>
<thead>
<tr>
<th>Function</th>
<th>Argument Definitions</th>
<th>Returns</th>
</tr>
</thead>
</table>
| LPAD (char_expression, length [, pad_expression ]) | text | Returns `char_expression` padded with optional `pad_expression` until the length of the resulting string is equal to argument `length`.
|                                |                       | • `char_expression` must evaluate to a CHAR value.  
|                                |                       | • `length` must evaluate to an INTEGER value.  
|                                |                       | • `pad_expression`, if specified, must evaluate to a CHAR value.  
|                                |                       | • If `pad_expression` is not specified, spaces are used for padding.  
|                                |                       | • If any expression is null LPAD returns NULL.  |
| LTRIM (char_expression [, char_set ]) | text | Removes leading characters in `char_expression`, present in `char_set` and returns the resultant string. Thus, the first character in the result is guaranteed not in `char_set`. If `char_set` is omitted, the function removes leading and trailing spaces from `char_expression`.
|                                |                       | • `char_expression` must evaluate to a CHAR value.  
|                                |                       | • `char_set`, if specified, must evaluate to a CHAR value.  |
| MATCH (char_expression, pattern_expression) | int | Compares `char_expression` to `pattern_expression`. MATCH returns 1 if the expression matches the pattern and zero if the expression does not match.
|                                |                       | • `char_expression` and `pattern_expression` must evaluate to CHAR values.  
|                                |                       | • Trailing blanks are significant.  
|                                |                       | • If any expression is null MATCH returns NULL.  
|                                |                       | The following special characters can be used in `pattern_expression`:
|                                |                       | • * (asterisk) : multi-character wildcard  
|                                |                       | • . (period) : single-character wildcard  
|                                |                       | • ~ (tilde) : escape character  |
| MATCHI (char_expression, pattern_expression) | int | Compares `char_expression` to `pattern_expression` as a case-insensitive comparison. MATCHI returns 1 if the expression matches the pattern and zero if the expression does not match.
|                                |                       | • `char_expression` and `pattern_expression` must evaluate to CHAR values.  
|                                |                       | • Trailing blanks are significant.  
|                                |                       | • If either expression is null MATCHI returns NULL.  
|                                |                       | The following special characters can be used in `pattern_expression`:
|                                |                       | • * (asterisk) : multi-character wildcard  
|                                |                       | • . (period) : single-character wildcard  
<p>|                                |                       | • ~ (tilde) : escape character  |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMCMP (char_expression1, char_expression2, count)</td>
<td>int</td>
<td>Compares count bytes of <code>char_expression1</code> and <code>char_expression2</code> and returns zero if both strings are equal, a negative value if <code>char_expression1</code> is less than <code>char_expression2</code>, or a positive value if <code>char_expression1</code> is greater than <code>char_expression2</code>.</td>
</tr>
<tr>
<td>REPEAT (char_expression, count)</td>
<td>int</td>
<td>Returns a character string composed of <code>char_expression</code> repeated <code>count</code> times.</td>
</tr>
<tr>
<td>RIGHT (char_expression, count)</td>
<td>text</td>
<td>Returns rightmost <code>count</code> of characters from <code>char_expression</code>.</td>
</tr>
<tr>
<td>ROWID (0, 0)</td>
<td>double</td>
<td>Returns the numeric row ID value if available.</td>
</tr>
<tr>
<td>RPAD (char_expression, length [ , pad_expression ] )</td>
<td>text</td>
<td>Returns <code>char_expression</code> padded with <code>pad_expression</code> such that after padding, length of result equals <code>length</code>.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Parameters</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>RTRIM (char_expression [, char_set ] )</td>
<td>Removes trailing characters in <code>char_expression</code> present in <code>char_set</code> and returns the resultant string. Thus, the last character in the result is guaranteed not in <code>char_set</code>. If <code>char_set</code> is omitted, the function removes trailing spaces from <code>char_expression</code>.</td>
<td><code>char_expression</code> must evaluate to a CHAR value. <code>char_set</code>, if specified, must evaluate to a CHAR value. If either argument is null <code>RTRIM</code> returns NULL.</td>
</tr>
<tr>
<td>SPACE (expression)</td>
<td>Returns a character string consisting of a number of spaces specified by <code>expression</code>.</td>
<td><code>expression</code> must evaluate to an exact numeric value. If <code>expression</code> is null <code>SPACE</code> returns NULL.</td>
</tr>
<tr>
<td>SUBSTRING (char_expression, start [, length ] )</td>
<td>Returns the substring of <code>char_expression</code> beginning at <code>start</code> and <code>length</code> characters long. If <code>length</code> is not specified, substring starting at <code>start</code> up to the end of <code>char_expression</code> is returned.</td>
<td><code>char_expression</code> must evaluate to a CHAR value. <code>start</code> must evaluate to an exact numeric value. The first character position in <code>char_expression</code> is 1. <code>length</code>, if specified, must evaluate to an exact numeric value. If any argument is null <code>SUBSTR</code> returns NULL.</td>
</tr>
<tr>
<td>UPPER (char_expression )</td>
<td>Returns <code>char_expression</code> after converting all characters to uppercase.</td>
<td><code>char_expression</code> must evaluate to a CHAR value. If <code>char_expression</code> is null <code>UPPER</code> returns NULL.</td>
</tr>
</tbody>
</table>
# Date and Time Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| **ADD_DAYS ( date_expression, integer_expression )** | DATE | Adds to the specified `date_expression` value the specified number of `integer_expression` days and returns the resultant date value.  
  * `date_expression` must be a DATE type value.  
  * `integer_expression` must be a numeric value.  
  * If either expression is NULL, result is NULL. |
| **ADD_MONTHS ( date_expression, integer_expression )** | DATE | Adds to the specified `date_expression` value the number of `integer_expression` months and returns the resultant date value.  
  * `date_expression` must be a DATE type value.  
  * `integer_expression` must be a numeric value.  
  * If either expression is NULL, result is NULL. |
| **CURDATE ( )** | DATE | Returns the current date as a DATE value. This function takes no arguments. |
| **CURTIME ( )** | TIME | Returns the current time as a TIME value. This function takes no arguments. |
| **DAYNAME ( date_expression )** | text | Returns a character string for the day name portion of `date_expression`. `date_expression` can be the name of a column, the result of another scalar function, or a DATE or TIMESTAMP literal. The return value is one of SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY or SATURDAY. |
| **DAYOFMONTH ( date_expression )** | int | Returns the day of month of `date_expression` as a short integer value in the range 1 - 31.  
  * The argument to the function must evaluate to a DATE type value.  
  * If `date_expression` is null DAYOFMONTH returns null. |
| **DAYOFWEEK ( date_expression )** | int | Returns the day of week of `date_expression` as an integer value in the range of 1 – 7, where 1 is Sunday, 2 is Monday, etc.  
  * `date_expression` must evaluate to a DATE type value.  
  * If `date_expression` is null DAYOFWEEK returns null. |
<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DAYOFYEAR</strong> (date_expression)</td>
<td>int</td>
<td>Returns the day of year of date_expression as an integer value in the range 1 - 366.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• date_expression must evaluate to a DATE type value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If date_expression is null DAYOFYEAR returns null.</td>
</tr>
<tr>
<td><strong>HOUR</strong> (time_expression)</td>
<td>int</td>
<td>Returns the hour of time_expression as an integer value in the range 0 - 23.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• time_expression must evaluate to a TIME value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If time_expression is null HOUR returns null.</td>
</tr>
<tr>
<td><strong>LAST_DAY</strong> (date_expression)</td>
<td>DATE</td>
<td>Returns the date corresponding to the last day of the month containing date_expression.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• date_expression must evaluate to a DATE value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If date_expression is null LAST_DAY returns null.</td>
</tr>
<tr>
<td><strong>MINUTE</strong> (time_expression)</td>
<td>int</td>
<td>Returns the minute value of time_expression as an integer in the range 0 - 59.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• time_expression must evaluate to a TIME value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If time_expression is null MINUTE returns null.</td>
</tr>
<tr>
<td><strong>MONTH</strong> (date_expression)</td>
<td>int</td>
<td>Returns the month number of date_expression as an integer value in the range 1 - 12.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• date_expression must evaluate to a DATE value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If date_expression is null, MONTH returns null.</td>
</tr>
<tr>
<td><strong>MONTHNAME</strong> (date_expression)</td>
<td>text</td>
<td>Returns the name of the month (for example, JANUARY, through DECEMBER) for the month portion of date_expression.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• date_expression must evaluate to a DATE value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If date_expression is null MONTHNAME returns null.</td>
</tr>
<tr>
<td><strong>NOW</strong> ()</td>
<td>TIMESTAMP</td>
<td>Returns the current date and time as a TIMESTAMP value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This function takes no arguments.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>QUARTER</strong> (date_expression)</td>
<td>int</td>
<td>Returns the quarter of date_expression as an integer value in the range 1 - 4.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• date_expression must evaluate to a DATE value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If date_expression is null QUARTER returns null.</td>
</tr>
<tr>
<td>Function</td>
<td>Return Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SECOND ( time_expression )</td>
<td>int</td>
<td>Returns the seconds <em>time_expression</em> as an integer value in the range 0 - 59.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <em>time_expression</em> must evaluate to a TIME value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <em>time_expression</em> is null SECOND return null.</td>
</tr>
<tr>
<td>SYSDATE ()</td>
<td>DATE</td>
<td>Returns the current date as a DATE value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This function takes no arguments, and trailing parentheses are optional.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This function is similar to CURDATE.</td>
</tr>
<tr>
<td>SYSTIME ()</td>
<td>TIME</td>
<td>Returns the current time as a TIME value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This function takes no arguments, and trailing parentheses are optional.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This function is similar to CURTIME.</td>
</tr>
<tr>
<td>SYSTIMESTAMP ()</td>
<td>TIMESTAMP</td>
<td>Returns the current date and time as a TIMESTAMP value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This function takes no arguments, and trailing parentheses are optional.</td>
</tr>
<tr>
<td>WEEK ( date_expression )</td>
<td>int</td>
<td>Returns the week of <em>date_expression</em> as an integer value in the range 1 - 53.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <em>date_expression</em> must evaluate to a DATE value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <em>date_expression</em> is null WEEK returns null.</td>
</tr>
<tr>
<td>YEAR ( date_expression )</td>
<td>int</td>
<td>Returns the year of <em>date_expression</em> as an integer value in the range 0 - 9999.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <em>date_expression</em> must evaluate to a DATE value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <em>date_expression</em> is null YEAR return null</td>
</tr>
</tbody>
</table>

### Mathematical Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS ( expression )</td>
<td>int</td>
<td>Computes absolute value of <em>expression</em>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The argument to the function must be of type TINYINT, SMALLINT, INTEGER, NUMERIC, REAL or FLOAT.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <em>expression</em> evaluates to null, the result is null.</td>
</tr>
<tr>
<td>Function</td>
<td>Return Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ACOS (expression)</td>
<td>double</td>
<td>Returns arccosine of expression. ACOS takes the ratio (expression) of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side adjacent to the angle divided by the length of the hypotenuse. The result is expressed in radians and is in the range -PI/2 to PI/2 radians. To convert degrees to radians, multiply degrees by PI/180. To convert radians to degrees, multiply radians by 180/PI.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• expression must be in the range -1 to 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• expression must evaluate to an approximate numeric data type.</td>
</tr>
<tr>
<td>ASIN (expression)</td>
<td>double</td>
<td>Returns the arcsine of expression. ASIN takes the ratio (expression) of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse. The result is expressed in radians and is in the range -PI/2 to PI/2 radians. To convert degrees to radians, multiply degrees by PI/180. To convert radians to degrees, multiply radians by 180/PI.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• expression must be in the range -1 to 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• expression must evaluate to an approximate numeric data type.</td>
</tr>
<tr>
<td>ATAN (expression)</td>
<td>double</td>
<td>Returns the arctangent of expression. ATAN takes the ratio (expression) of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle. The result is expressed in radians and is in the range -PI/2 to PI/2 radians. To convert degrees to radians, multiply degrees by PI/180. To convert radians to degrees, multiply radians by 180/PI.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• expression must be in the range -1 to 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• expression must evaluate to an approximate numeric data type.</td>
</tr>
<tr>
<td>CABS (expression)</td>
<td>int</td>
<td>Calculates the absolute value of a complex number.</td>
</tr>
<tr>
<td>CEIL (expression)</td>
<td>double</td>
<td>Returns the smallest integer greater than or equal to expression. CEIL is similar to CEILING function.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• expression must evaluate to a numeric type.</td>
</tr>
<tr>
<td>CEILING (expression)</td>
<td>double</td>
<td>Returns the smallest integer greater than or equal to expression. CEILINGING is similar to CEIL function.</td>
</tr>
<tr>
<td>Function</td>
<td>Return Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| COS (expression) | double      | Returns the cosine of expression. COS takes an angle (expression) and returns the ratio of two sides of a right triangle. The ratio is the length of the side adjacent to the angle divided by the length of the hypotenuse. To convert degrees to radians, multiply degrees by Pi/180. To convert radians to degrees, multiply radians by 180/Pi.  
  - expression specifies an angle in radians.  
  - expression must evaluate to an approximate numeric data type. |
| EXP (expression) | double      | Returns the exponential value of expression (e raised to the power of expression).  
  - expression must evaluate to an approximate numeric data type.  
  - If expression is null EXP returns null. |
| FABS (expression) | double      | Returns the absolute value of expression.  
  - expression must evaluate to a numeric value.  
  - If expression is null FABS returns null. |
| FLOOR (expression) | int         | Returns the largest integer less than or equal to expression.  
  - expression must evaluate to a numeric value.  
  - If expression is null FLOOR returns null. |
| FMOD (expression1, expression2) | double | Calculates expression1 modulo expression2, the remainder of expression1 / expression2.  
  - Both expressions must evaluate to a numeric value.  
  - If either expression is null FMOD returns null. |
| LABS (expression) | integer     | Computes the absolute value of an INTEGER value.  
  - expression must evaluate to an INTEGER value.  
  - If expression is null LABS returns null. |
| LOG10(expression) | double      | Returns the base 10 logarithm of expression.  
  - expression must evaluate to a NUMERIC value.  
  - If expression is null LOG10 returns null. |
| MOD (expression1, expression2) | int        | Returns the remainder of expression1 divided by expression2.  
  - expression1 and expression2 must evaluate to exact numeric values.  
  - If expression2 evaluate to zero, a runtime error will be generated, which will cause the expression being evaluated as false.  
  - If either expression is null MOD returns null. |
| PI ()            | number      | Returns the constant value of pi as an approximated numeric value. This function takes no arguments. |
### POWER( expression1, expression2 )

| Returns | expression1 raised to the power of expression2.
|---------|--------------------------------------------------
|        | • Both expression1 and expression2 must evaluate to numeric values.
|        | • If either expression is null POWER returns null.

### RAND( [ expression ] )

| Returns | a randomly-generated number, using expression as an optional seed value.
|---------|--------------------------------------------------
|        | • expression, if specified, must evaluate to an exact numeric value.
|        | • If expression is specified and it is null RAND returns null.

### SIGN ( expression )

| Returns | +1 if expression > 0
|---------|--------------------------------------------------
|        | -1 if expression < 0
|        | 0 if expression = 0
|        | • expression must evaluate to a numeric value.
|        | • If expression is null SIGN returns null.

### SIN ( expression )

| SIN takes an angle (expression) and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse.
| Returns: |
|         | To convert degrees to radians, multiply degrees by Pi/180. To convert radians to degrees, multiply radians by 180/Pi.
|         | • expression must evaluate to a numeric value.
|         | • If expression is null SIN returns null.

### SQRT ( expression )

| SQRT return the square root of expression.
| expression must evaluate to a numeric value.
| If expression is null SQRT returns null.

### TAN ( expression )

| TAN returns the tangent of expression.
| TAN takes an angle (expression) and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.
| Returns: |
|         | To convert degrees to radians, multiply degrees by Pi/180. To convert radians to degrees, multiply radians by 180/Pi.
|         | • expression must evaluate to a numeric value.
|         | • If expression is null TAN returns null.

### Type Casting

The filter expression parser allows you to use explicit type casts in expressions. This is very useful if you are comparing fields of different types and want to control the result of an expression.

For example, suppose “Salary” is a CT_MONEY field and “Average” is a CT_DFLOAT field; type casts can be used as illustrated in the following expression: (Salary - (int)Average) > 500
The following type casts may be used in conditional expressions:

- (int) or (long): Convert the result of expression to integer (32 bit).
- (unsigned [int | long]): Convert the result of expression to unsigned integer (32 bit).
- (double): Convert the result of expression to double.

You cannot type cast a string expression.

**Automatic Type Promotion**

When mixing different types in an expression without explicit type casting, the conditional expression parser automatically promotes the types using the following rule:

1. signed and unsigned integers - promoted to unsigned integer (64-bit)
2. signed integer and double - promoted to double
3. unsigned integer and double - promoted to double

In the great majority of cases, mixing strings with numeric values returns a parsing error.

**Operators**

The following operators are allowed in filters and conditional expressions.

### Mathematical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Adds two operands</td>
</tr>
<tr>
<td>-</td>
<td>Subtracts two operands or negates an operand (e.g., -5)</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>%</td>
<td>Modulus</td>
</tr>
</tbody>
</table>

### Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
</tbody>
</table>

### Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>And</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Not</td>
</tr>
</tbody>
</table>
Binary Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>And</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>~</td>
<td>Not</td>
</tr>
<tr>
<td>^</td>
<td>Xor</td>
</tr>
</tbody>
</table>

NULL Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>IS NULL</td>
<td></td>
</tr>
<tr>
<td>NOT NULL</td>
<td></td>
</tr>
</tbody>
</table>

Custom Application Expressions

Use the conditional expression parser/analyzer to evaluate application specific expressions. The c-treeACE expression parser is a full standalone feature in and of itself, and can be used directly in your own applications for purposes other than data filters, conditional indices or partitioned files. Two core API functions provide a powerful interface into this advanced expression handling.

c-treeACE’s expression parser/analyzer requires two key steps:

1. Call `cndxparse()` to parse your expression, producing an expression tree that the expression analyzer later evaluates.
2. Call `cndxeval()` to evaluate your expression tree using data from a buffer in memory.

For a complete sample program, see `ctexpr.c` in the `ctree/source` directory.

See also:
- Parsing Expressions (page 138)
- Evaluating Expressions (page 140)

Parsing Expressions

Parsing your expression involves three steps:

1. Define a DODA structure.
2. Parse the DODA into a record schema and field name list.
3. Parse your expression to produce an expression tree.

Sample code to perform these steps is shown below. This code assumes c-treeACE has been initialized prior to calling `ctparsedoda()`.

Expression Parsing Example

```c
#include "ctcndx.h" /* For PTREE type */
/* Define a DODA structure. */
DATOBJ doda[] = {
   "CustomerNumber", 0, CT_INT4U),
```

```
To invoke the expression parser to parse a string expression and produce an expression tree, call `cndxparse()`, which is declared as follows:

```c
PTREE cndxparse(pConvMap Schema, pTEXT Names, pTEXT InputText, NINT InputTextSize)
```

Schema is a pointer to a record schema derived from a DODA definition. Names is a pointer to a list of the field names from the DODA. `InputText` points to a NULL-terminated string expression, and `InputTextSize` is the length of `InputText`.

One of the most useful features of the expression parser is its ability to associate symbolic names in expressions with data in a buffer in memory. In order to use this ability, you must define a record schema, known as a DODA (Data Object Definition Array). A DODA is an array of field specifications, each of which contains a field name, field offset, field type, and a field length. By providing the expression parser with a DODA, you may include references to DODA field names in your expressions. See the sample code shown later in this section for an example of a DODA definition.

While a DODA is conveniently defined in your application using an array of DATOBJ structures, `cndxparse()` does not take a DODA in DATOBJ form, but instead accepts a record schema and a list of the field names from the DODA. In order to simplify converting your DODA into the required record schema and field name list, FairCom has written a utility function, `ctparsedoda()`. This function can be found in the sample file `ctexpr.c`.

`ctparsedoda()` is declared as follows:

```c
COUNT ctparsedoda(pDATOBJ doda, UCOUNT numfld, ppTEXT ppschema, ppTEXT ppnames)
```
Evaluating Expressions

Having produced an expression tree, you are ready to evaluate the expression using the expression analyzer. To evaluate your expression, call `cndxeval()`, which is declared as follows:

```c
COUNT cndxeval(PTREE Tree, pVOID Recptr, pConvMap Schema)
```

- `Tree` is an expression tree returned by `cndxparse()`.
- `Recptr` points to a buffer containing data you wish to evaluate with your expression.
- `Schema` is the record schema `ctparsedoda()` produced from your DODA definition. The record schema is used to associate data in `Recptr` with field names specified in your expression.

**Note:** Before you call `cndxeval()` the first time, you must ensure that a run-time stack has been allocated for the expression analyzer.

To summarize, evaluating your expression involves three steps:

1. Allocate a run-time stack for the expression analyzer (first time only).
2. Set up a buffer containing the field data used in the expression.
3. Evaluate the expression.

If you wish, you can repeat steps 2) and 3) multiple times. Sample code to perform these steps is shown below. It is assumed the `Get_Buffer()` routine allocates a record buffer and initializes it with data conforming to the field definitions specified in the DODA.

**Expression Evaluation Example**

```c
COUNT retcidx; /* Result of expression evaluation. */
pTEXT recbuf; /* Record buffer. */

/* Allocate a run-time stack for the expression analyzer (first time only). */
if (!ctcidxStk) {
    ctcidxStk = (pVOID) getcndxmem(CNDX_MAX_STACK * ctSIZE(PLEAF));
    if (!ctcidxStk) {
        printf("Unable to allocate memory for run-time stack.\n");
        ctrt_exit(1);
    }
}

/* Set up a buffer containing the field data used in the expression. */
Get_Buffer(&recbuf);

/* Evaluate the expression. */
retcidx = cndxeval(ptree, recbuf, {pConvMap}schema);
if (retcidx<0)
    printf("The expression cannot be evaluated for this record
            - error %d.\n", uerr_cod);
else if (retcidx)
    printf("The expression evaluates to TRUE for this record.\n");
else
    printf("The expression evaluates to FALSE for this record.\n");
```

Remember to always free any memory used by your record schema, field name list, and expression tree when you are finished with them. Use the following c-treeACE functions to do so:

- `mbfree(schema)`
- `mbfree(names)`
- `cndxfree(ptree)`
Variable-length Records with Conditional Expressions

When using data filters or conditional indices with variable length data files, a record retrieval that does not bring back the entire record will return a **CVAL_ERR** error (598), which indicates the expression could not be evaluated.

There are two types of retrievals that result in less than the entire record being read:

1. Calling the fixed-length versions of the ISAM routines such as **FirstRecord()** or **NextRecord()** instead of **FirstVRecord()** or **NextVRecord()**. The fixed-length calls cannot be used to read variable-length records with a data filter or conditional index.

2. Calling the variable-length versions of the ISAM routines with a buffer length insufficient to hold the entire record.

When an ISAM call fails (with **CVAL_ERR** (598) or some other error), the current ISAM position is NOT updated. Therefore the following pseudo-code sequence will NOT work because the **FirstRecord()** did not establish the failing record as the current ISAM position, and the **GETVLEN()** call would reference the record at the current ISAM position before the **FirstRecord()** call:

```
SETFLTR(datno,...);
if (FRSREC(...) == CVAL_ERR) {
  vlen= GETVLEN(datno);
  rc= FRSVREC(...,&vlen);
}
```

Using the variable-length versions of the ISAM routines provides a workable approach. The following pseudo-code works, with one proviso - the subsequent calls to the ISAM routine can also fail with a **CVAL_ERR** (598) because they may have skipped forward to an even larger record:

```
SETFLTR(datno,...);
oldlen = vlen;
if (FRSVREC(...,bufr,&vlen) == CVAL_ERR && oldlen < vlen) {
  free(bufr);
  oldlen = vlen;
  bufr   = calloc(vlen);
  rc     = FRSVREC(...,bufr,&vlen);
}
```

The second call to **FirstVRecord()** could also return the **CVAL_ERR** (598) because while the record that originally caused the **CVAL_ERR** (598) can now be read completely, if it failed the filter, the next records will be read automatically until a record is found that passes the filter; but these subsequent reads can also encounter a record that is bigger than the new buffer size.

The following pseudo-code loop should work with any of the variable length versions of the ISAM calls:

```
SETFLTR(datno,...);
oldlen = vlen;
while CVAL_ERR == (xyzVREC(...,bufr,&vlen) && oldlen < vlen) {
  free(bufr);
  oldlen = vlen;
  bufr   = calloc(vlen);
  rc     = FRSVREC(...,bufr,&vlen);
}
```

if (isam_err)
If one knows ahead of time that there is a maximum record length for the file, then simply using a buffer of this known maximum size eliminates the need to loop over the **CVAL_ERR** (598) caused by an insufficient buffer size.

### Conditional Expression Callback Function

c-treeACE provides a developer-defined callback function to perform additional advanced custom data filtering and conditional index evaluation. Instead of calling c-tree’s internal expression evaluator to analyze criteria, a user-defined function is called, allowing advanced filtering and control with application specific code.

#### Callback Functions

The module **ctcblk.c** contains the routines declared below. **ctfiltercb()** is called when a user-defined filter must be evaluated. Whenever a file open retrieves stored conditional index callback expressions, when a conditional index callback expression is created, or when a **SetDataFilter()** callback expression is created, the **ctfiltercb_init()** routine is called with a pointer to the callback expression. Whenever a file is closed, or a **SetDataFilter()** is cleared, the **ctfiltercb_uninit()** routine is called for each callback expression.

**ctfiltercb_init()** is expected to return zero (0) on error and non-zero on initialization, though at this point the error return is ignored. **ctfiltercb_init()** should set a state variable so the **ctfiltercb_uninit()** knows whether the init was called for this particular callback filter.

**ctfiltercb_uninit()** is expected to return zero (0) on error and non-zero on uninitialization, though at this point the error return is ignored. **ctfiltercb_uninit()** should check a state variable to determine whether or not **ctfiltercb_init()** was called for this particular callback filter.

Stub functions with simple debug print statements are part of the base distribution. A developer taking advantage of expression callback routines must adapt these callback functions to their particular requirements.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pTEXT Clbk</strong></td>
<td>Pointer to a NULL terminated ASCII string beginning with the <strong>ctCNDXclbkCHR</strong> (@) character. Presumably, the string starting in the 2nd position (i.e., Clbk + 1) points to a callback function designator.</td>
</tr>
<tr>
<td><strong>pVOID Recptr</strong></td>
<td>Pointer to a record image</td>
</tr>
<tr>
<td><strong>pConvMap Schema</strong></td>
<td>Pointer to a record schema</td>
</tr>
<tr>
<td><strong>VRLEN fixlen</strong></td>
<td>Fixed length of the record image</td>
</tr>
<tr>
<td><strong>VRLEN datlen</strong></td>
<td>Full length of the record image</td>
</tr>
<tr>
<td><strong>pinHan</strong></td>
<td>A macro the converts to <strong>ctWNGV</strong> for standalone or client libraries defining <strong>ctNOGLOBALS</strong> or <strong>lctgv</strong> for Server or Bound Server code.</td>
</tr>
</tbody>
</table>
Callback Expression String

As described above, $Clbk$ points to an ASCII string beginning with the "at" sign '@'. The string following '@' is completely arbitrary, and is interpreted to determine what type of callback routine is intended, assuming more than one type of callback is required by the application. A simple scheme would use callback strings with a unique character in the second position (following '@'). This permits a simple switch statement routing callbacks to desired code, as shown in the following pseudo-code:

```c
switch (*(Clbk + 1)) {
    case 'C':
        do the Customer Number check
        break;
    case 'Z':
        do the Zero Balance check
        break;
    case 'T':
        do the Total Funds check
        break;
    default:
        set the return code to -CVAL_ERR to indicate the
        filter could not be evaluated
        break;
}
```

In this example scheme, the only significant portion of the callback designator is the first character after the @.

Record Batches

There are situations where a record fetch, a record insert or a record delete operation has to be applied to a group of related records. For example, if an invoice record is being displayed, it may be necessary to retrieve all invoice items for that invoice as well. The following code fragment shows a typical c-treeDB C function to extract all items of an invoice, given the invoice number.

c-treeDB C API Example

```c
void GetInvoiceItems(CTHANDLE hRecord, NINT Invoice) {
    NINT count = 0;
    TEXT target[32];
    VRLEN len = sizeof(target);

    /* find the first record that match the Invoice */
    ctdbClearRecord(hRecord);
    ctdbSetFieldAsSigned(hRecord, 0, Invoice);
    if (ctdbFindRecord(hRecord, CTFIND_GE) == CTDBRET_OK) {
        LONG val;
        /* make sure all record invoice numbers match */
        ctdbGetFieldAsSigned(hRecord, 0, &val);
```
while (val == (LONG)Invoice)
{
    val = -1;
    count++;
    if (ctdbNextRecord(hRecord) == CTDBRET_OK)
        ctdbGetFieldAsSigned(hRecord, 0, &val);
}
printf("%d records found\n", count);
}

The code fragment above shows how easy it is to use c-treeDB functionality to retrieve a number of related records from a table. A close inspection of the code reveals that while it is simple to implement, it does have a hidden issue that may affect the overall performance of the system, specially when running in client/server mode: for every record that is fetched, the client must place a request to the server, which must travel the network, the c-tree Server must interpret the client request, perform the record retrieval operation and return the record to the client, again traveling across the network.

It is obvious the operation described above would be more efficient if we could, in just one call, request the c-tree Server to perform a given operation on a group of related records, and return all the resulting records, or as many records as possible, in one client request.

The ability of perform record retrievals, insert or delete operations on a group of related records is implemented under c-treeDB as record batch operations. The same code fragment above implemented with c-treeDB C API batch record facility would look as follows:

c-treeDB C API Batch Example

```c
void GetInvoiceItems(CTHANDLE hRecord, NINT Invoice)
{
    NINT count = 0;

    /* set the partial target key */
    ctdbClearRecord(hRecord);
    ctdbSetFieldAsSigned(hRecord, 0, Invoice);
    /* set the batch operation */
    if (ctdbSetBatch(hRecord, CTBATCH_GET, sizeof(Invoice), 0) == CTDBRET_OK)
    {
        /* retrieve records */
        while (ctdbNextBatch(hRecord) == CTDBRET_OK)
            count++;
        /* terminate batch operations */
        ctdbEndBatch(hRecord);
    }
    printf("%d records found\n", count);
}
```

The code fragment above is even simpler than the initial example, and it also performs far fewer c-tree Server calls to perform the operations. c-treeDB batch record functionality is implemented on top of c-tree Plus ISAM batch operations.

**Batches with c-treeDB**

The following types of batch operations may be performed:
Retrieve records by partial key
All records with keys matching a partial target key are loaded into an internally maintained buffer region. If there are more records than fit in the buffer, those that fit are loaded, and subsequent calls will retrieve the remaining records.

Retrieve records by index range
Retrieve all records that match an index range expression. All matched records are loaded into an internally maintained buffer region. If there are more records than fit in the buffer, those that fit are loaded, and a subsequent call will retrieve the remaining records.

Retrieve records by physical order
All records of a table are loaded by physical order into an internally maintained buffer region. If the selected records do not fit in the buffer, those that fit are loaded, and subsequent calls will retrieve the remaining records.

Insert a group of records
A group of new records are loaded into an internally maintained buffer region and this group of records are inserted into a table.

Delete a group of records
All records with a key matching a partial target key are deleted.

Batch Modes
You must specify one of the following Mandatory modes when calling the ctdbSetBatch() function or the CTRecord::SetBatch() method:

<table>
<thead>
<tr>
<th>MODE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTBATCH_GET</td>
<td>Retrieve a group of related records by partial key</td>
</tr>
<tr>
<td>CTBATCH_RANGE</td>
<td>Retrieve a group of related records based on an index range expression</td>
</tr>
<tr>
<td>CTBATCH_PHYS</td>
<td>Retrieve records from a table in physical order. The starting record for the batch retrieval may be specified. (A physical order batch read is slightly faster than going through an index as it has fewer reads to do.)</td>
</tr>
<tr>
<td>CTBATCH_DEL</td>
<td>Delete a group of related records by partial key</td>
</tr>
<tr>
<td>CTBATCH_INS</td>
<td>Insert a group of records</td>
</tr>
</tbody>
</table>

The following modes are optional and can be OR-ed to the mandatory mode to specify other details on how the batch operation is to be performed.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTBATCH_GKEY</td>
<td>Process records with a greater than or equal key match with the target key. When this mode is specified, the number of matched records is not readily available. ctdbBatchLocked() and CTRecord::BatchLocked returns a value one greater than ctdbBatchLoaded() to indicate there may be more records to process. This mode is applicable only with CTBATCH_GET and CTBATCH_DEL modes and cannot be used with CTBATCH_LKEY.</td>
</tr>
</tbody>
</table>
### Mode | Description
--- | ---
CTBATCH_LKEY | Process records that have a less than or equal key match with the target key. This mode is applicable only with CTBATCH_GET and CTBATCH_DEL modes and can not be used with CTBATCH_GKEY.
CTBATCH_VERIFY | Verify that the keys in the index match the values in the key fields of the record.
CTBATCH_LOCK_KEEP | Keep all records locked after ...EndBatch() is called. Without this mode, all records locks are released when ...EndBatch() is called. This option is only in effect when used with CTBATCH_LOCK_READ or CTBATCH_LOCK_WRITE.
CTBATCH_LOCK_READ | Place a read lock on each record that matches the partial key.
CTBATCH_LOCK_WRITE | Place a write lock on each record that matches the partial key.
CTBATCH_LOCK_BLOCK | Convert a CTBATCH_LOCK_READ or CTBATCH_LOCK_WRITE to blocking read and blocking write locks, respectively.
CTBATCH_LOCK_ONE | Implement an alternative locking strategy: only locks the record during the record read; original locking strategy keeps locks on during entire batch processing.
CTBATCH_COMPLETE | ...SetBatch() returns a success code only if all matching records are successfully locked. You must specify either CTBATCH_LOCK_READ or CTBATCH_LOCK_WRITE.

### Starting a new batch operation
The way a new batch operation is started will depend on the type of batch operation you are performing:
- Retrieving records by partial key
- Retrieving records by index range
- Retrieving records by physical order
- Deleting a group of records
- Inserting a group of records

Each of these situations is discussed below.

#### Retrieving records by partial key
All records with key matching a partial target key are loaded into a buffer region maintained internally by c-treeDB. If the selected records do not fit in the buffer, those that fit are loaded, and subsequent calls will retrieve the remaining records.

The following steps must be taken to perform a batch retrieval operation based on a partial key:

1. **Clear a record buffer by calling the CTRecord::Clear() method.**
2. **Use CTRecord::SetFieldAs...() methods to set the fields that form the partial target key that will be used to select a group of records.**
3. **Call the CTRecord::SetBatch() method, with the CTBATCH_GET mode, to start a new record retrieval batch operation.**
4. Call the `CTRecord::NextBatch()` method repeatedly until all related records are retrieved. `CTRecord::NextBatch()` returns false to indicate no more records are available.

5. When you are done with the batch records, call the `CTRecord::EndBatch()` method to terminate the batch operation. Please note that another batch operation can only start after the current batch operation is terminated.

To start a new batch operation to retrieve records, you must first establish the partial target key that will identify the group of related records. This is accomplished by clearing a record buffer and setting the fields that form the partial target key. For example, if an invoice item table has one index with segments based on invoice number and item number fields, the following partial code will create a partial target key that will select all invoice item records associated with a particular invoice number:

**Example**

```c++
// set the partial target key
hRecord.Clear();
hRecord.SetFieldAsSigned("Invoice", Invoice);
```

After preparing the partial target key, a new batch operation is started by calling the `CTRecord::SetBatch()` method. Continuing from the example above, a new batch operation is started by performing the following call:

**Example**

```c++
try {
    // set the batch operation
    hRecord.SetBatch(CTBATCH_GET, sizeof(Invoice), 0);
    // retrieve and display all records
    while (hRecord.NextBatch())
        PrintRecord(hRecord);
    // terminate batch operations
    hRecord.EndBatch();
} catch (CTException &err) {
    printf("Batch operation failed with error %d\n", err.GetErrorCode());
}
```

`CTRecord::SetBatch()` takes as first argument a valid record handle. The second argument is the batch mode that will direct how the batch operation will be performed. The chapters below describe the available batch modes in detail.

You must provide the length of the `SetBatch()`’s target key in bytes. The length of the target key will indicate how many bytes of the target key should be used to create the partial target key. The last parameter is the size of the buffer used internally by c-treeDB to handle batch operations. A zero value for the last parameter is an indication that the default value size should be used. The default buffer size is calculated as the size of the fixed portion of the record multiplied by 128.

**Retrieving records by index range**

All records that match an index range expression are loaded into a buffer region maintained internally by c-treeDB. If the selected records do not fit in the buffer, those that fit are loaded, and subsequent calls will retrieve the remaining records.
The following steps must be taken to perform an index range batch retrieval of records:

1. Establish an index range by calling the `CTRecord::RangeOn()` method.
2. Call the `CTRecord::SetBatch()` method with the `CTBATCH_RANGE` mode to start a new record retrieval batch operation.
3. Call the `CTRecord::NextBatch()` method repeatedly until all related records are retrieved. `CTRecord::NextBatch()` returns false to indicate no more records are available.
4. When you are done with the batch records, call the `CTRecord::EndBatch()` method to terminate the batch operation.
5. Call the `CTRecord::RangeOff()` method to terminate index range operations.

To start a new batch operation to retrieve records, you must first establish the index range operation that will identify the group of related records.

**Example:**

```c
try
{
    // build target key to be used in index range
    TEXT lRange[32];
    VRLEN len = (VRLEN)sizeof(lRange);
    NINT op = CTIX_EQ;

    hRecord.Clear();
    hRecord.SetFieldAsSigned("Invoice", Invoice);
    hRecord.BuildTargetKey(CTFIND_EQ, (pVOID)lRange, &len);

    // set the index range based on the target key
    hRecord.RangeOn(1, (pVOID)lRange, NULL, CTIX_EQ&op);
}
catch (CTException &err)
{
    printf("Index range operation failed with error %d\n", err.GetErrorCode());
}
```

After setting the index range operation, you may start a new batch operation by calling the `CTRecord::SetBatch()` method.

**Example:**

```c
try
{
    // set the batch operation
    hRecord.SetBatch(CTBATCH_RANGE, 0, 0);

    // retrieve and display all records
    while (hRecord.NextBatch())
    {
        PrintRecord(hRecord);
    }

    // terminate the batch operations
    hRecord.EndBatch();

    // terminate the index range
    hRecord.RangeOff();
}
catch (CTException &err)
{
    printf("Batch operation failed with error %d\n", err.GetErrorCode());
}
```
Notice when retrieving records by index range `SetBatch()`'s `targetLen` parameter is set to zero. The last parameter is the size of the buffer used internally by c-treeDB to handle batch operations. A zero value for the last parameter is an indication that the default value size should be used. The default buffer size is calculated as the size of the fixed portion of the record multiplied by 128.

**Retrieving records by physical order**

All records of a table are loaded by physical order into a buffer region maintained internally by c-treeDB. If the selected records do not fit in the buffer, those that fit are loaded, and subsequent calls will retrieve the remaining records.

The following steps must be taken to perform a physical order batch retrieval of records:

1. Call the `CTRecord:::SetBatch()` method with `CTBATCH_PHYS` mode to start a new record retrieval batch operation.
2. Continue to call the `CTRecord:::NextBatch()` method repeatedly until all related records are retrieved. `CTRecord:::NextBatch()` returns false to indicate no more records are available.
3. When you are done with the batch records, call the `CTRecord:::EndBatch()` method to terminate the batch operation.

**Example**

```
try
{
    // set the batch operation */
    hRecord.SetBatch(CTBATCH_PHYS, 0, 0);

    // retrieve and display all records
    while (hRecord.NextBatch())
        PrintRecord(hRecord);

    // terminate batch operations
    hRecord.EndBatch();
}
```

Notice when retrieving records by index range `SetBatch()`'s `targetLen` parameter is set to zero since no partial key is need for this operation. The last parameter is the size of the buffer used internally by c-treeDB code to handle batch operations. A zero value for this parameter is an indication that the default value size should be used. The default buffer size is calculated as the size of the fixed portion of the record multiplied by 128.

**Deleting a group of records**

If the intended batch operation is to delete a group of selected records, you need to initially set the partial target key to select the group of related records and then start the batch operation to delete the selected records.

Even if no records are retrieved with the delete operation, `CTRecord:::EndBatch()` must be called to terminate the current batch operation.

The following steps must be taken to perform a batch delete record operation:
1. Clear a record buffer by calling the \texttt{CTRecord::Clear()} method.
2. Use the \texttt{CTRecord::SetFieldAs...()} methods to set the fields that form the partial target key that will be used to select a group of records.
3. Call the \texttt{CTRecord::SetBatch()} method with the \texttt{CTBATCH_DEL} mode to delete a group of related records.
4. Call the \texttt{CTRecord::EndBatch()} method to terminate the delete record batch operation.

\textbf{Example}

\begin{verbatim}
try {
    // set the partial target key
    hRecord.Clearh();
    hRecord.SetFieldAsSigned("Invoice", Invoice);

    // set the batch operation
    hRecord.SetBatch(CTBATCH_DEL, sizeof(Invoice), 0);

    // end the batch operation
    hRecord.EndBatch();
}
catch (CTException &err) {
    printf("Batch operation failed with error %d\n", err.GetErrorCode());
}
\end{verbatim}

You must provide the length of \texttt{SetBatch()}’s Target key in bytes. The length of the target key indicates how many bytes of the target key should be used to create the partial target key. The last parameter is the size of the buffer used internally by c-treeDB to handle batch operations. A zero value for the last parameter indicates the default value size should be used. The default buffer size is calculated as the size of the fixed portion of the record multiplied by 128.

\textbf{Inserting a group of records}

A group of new records are loaded into a buffer region maintained internally by c-treeDB and this group of records are inserted into a table.

When the batch buffer fills up, the group of records stored in the batch buffer are inserted into the table. If \texttt{CTRecord::EndBatch()} is called and the batch buffer still contains records, a new insert record operation is performed for the remaining records before the batch operation is terminated.

For transaction controlled files, the batch insertion operation is treated as one all or nothing operation. If no explicit transaction is started, each insertion of records will start and end its own transaction. Even if an explicit transaction is started, each insertion operation is treated independently through safe points.

Currently, all records insertion operations do not perform any conversion of record images, key values and record position for heterogeneous client/server implementations.

The following steps must be taken to perform a batch insert record operation:

1. Call the \texttt{CTRecord::SetBatch()} method, with \texttt{CTBATCH_INS} mode, to insert a group of records.
2. For each record to be inserted perform the following operations:
   a. Call the \texttt{CTRecord::Clear()} method to clear a record buffer.
   b. For each field in the record call one of the \texttt{CTRecord::SetFieldAs...()} methods to set the field data.
c. Call the `CTRecord::InsertBatch()` method to insert the record into the batch buffer.

3. Call the `CTRecord::EndBatch()` method to indicate no more records will be inserted.

**Example**

```c
try {
    // set the batch operation
    hRecord.SetBatch(CTBATCH_INS, 0, 0);

    // prepare the first record
    hRecord.Clear();
    hRecord.SetFieldAsSigned("Invoice", Invoice); // invoice number
    hRecord.SetFieldAsSigned("ItemNbr", 1); // invoice item number
    hRecord.SetFieldAsSigned("Quantity", 100); // item quantity
    hRecord.SetFieldAsSigned("ItemCode", 1001); // item code
    hRecord.InsertBatch(); // insert record in batch

    // prepare the second record
    hRecord.Clear();
    hRecord.SetFieldAsSigned("Invoice", Invoice); // invoice number
    hRecord.SetFieldAsSigned("ItemNbr", 2); // invoice item number
    hRecord.SetFieldAsSigned("Quantity", 200); // item quantity
    hRecord.SetFieldAsSigned("ItemCode", 1002); // item code
    hRecord.InsertBatch(); // insert record in batch

    // terminate the batch operation
    hRecord.EndBatch();
} catch (CTException &err) {
    printf("Batch operation failed with error %d\n", err.GetErrorCode());
}
```

Notice when inserting a group of records `SetBatch()`'s `targetLen` parameter is set to zero as no partial key is needed for this operation. The last parameter is the size of the buffer used internally by c-treeDB to handle batch operations. A zero value for this parameter indicates the default value size should be used. The default buffer size is calculated as the size of the fixed portion of the record multiplied by 128.

**Retrieving records**

If the mode of the batch operation is one of `CTBATCH_GET`, `CTBATCH_RANGE` or `CTBATCH_PHYS` then it may be necessary to retrieve all records that match the batch criteria. The records are retrieved by calling the `CTRecord::NextBatch()` method.

`CTRecordNextBatch()` retrieves record data from the batch buffer maintained by c-treeDB's record handle. After a successful call to `CTRecordNextBatch()` the field data can be retrieved by calling the appropriate `CTRecord::GetFieldAs...()` functions.

**Example**

```c
try {
    // retrieve records
    while (CTRecord::NextBatch()) {
        CTString invoice = hRecord.GetFieldAsString("Invoice");
        CTString item = hRecord.GetFieldAsString("ItemNbr");
```
printf("%-11s %s\n", invoice.c_str(), item.c_str());
}
catch (CTException &err)
{
    printf("Batch operation failed with error %d\n", err.GetErrorCode());
}

Terminating a batch operation

A batch operation must be terminated by calling the `CTRecord::EndBatch()` method. Once a batch operation is started, by calling the `CTRecord::SetBatch()` method no other batch operation is allowed to start until the current batch operation is terminated.

Example

```cpp
try
{
    // set the partial target key
    hRecord.Clear();
    hRecord.SetFieldAsSigned("Invoice", Invoice);

    // set the batch operation
    hRecord.SetBatch(CTBATCH_DEL, sizeof(Invoice), 0);

    // end the batch operation
    hRecord.EndBatch();
}
catch (CTException &err)
{
    printf("Batch operation failed with error %d\n", err.GetErrorCode());
}
```

When performing batch retrieval operations, you may cancel the batch operation before retrieving all the records by calling the `CTRecord::EndBatch()` method.

If the batch operation is a `CTBATCH_RANGE` then you must also call the `CTRecord::RangeOff()` method to terminate the index range used for the batch operation.

Retrieving batch properties

Once a batch operation is started, the following batch properties can be retrieved by calling the appropriate functions or methods.

<table>
<thead>
<tr>
<th>c-treeDB C++ API Methods</th>
<th>Batch Properties Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CTRecord::BatchTotal</code></td>
<td>Retrieves the total number of records matching the batch criteria.</td>
</tr>
<tr>
<td><code>CTRecord::BatchLocked</code></td>
<td>Retrieves the number of records locked by a batch operation.</td>
</tr>
<tr>
<td><code>CTRecord::BatchLoaded</code></td>
<td>Retrieves the number of records loaded in a batch buffer maintained by the c-treeDB record handle.</td>
</tr>
<tr>
<td><code>CTRecord::BatchMode</code></td>
<td>Retrieves the batch mode set by the <code>CTRecord::SetBatch()</code> method.</td>
</tr>
<tr>
<td><code>CTRecord::IsBatchActive</code></td>
<td>Returns YES if a batch operation is active.</td>
</tr>
</tbody>
</table>
Reading and writing field data to a record buffer

The c-treeDB `CTRecord` class has the following methods to read field data from the record buffer:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetFieldAsBool</td>
<td>Read field data as Boolean value</td>
</tr>
<tr>
<td>GetFieldAsSigned</td>
<td>Read field data as signed integer value</td>
</tr>
<tr>
<td>GetFieldAsUnsigned</td>
<td>Read field data as unsigned integer value</td>
</tr>
<tr>
<td>GetFieldAsDate</td>
<td>Read field data as date value</td>
</tr>
<tr>
<td>GetFieldAsTime</td>
<td>Read field data as time value</td>
</tr>
<tr>
<td>GetFieldAsMoney</td>
<td>Read field data as money value</td>
</tr>
<tr>
<td>GetFieldAsFloat</td>
<td>Read field data as double value</td>
</tr>
<tr>
<td>GetFieldAsDateTime</td>
<td>Read field data as time stamp value</td>
</tr>
<tr>
<td>GetFieldAsString</td>
<td>Read field data as string value</td>
</tr>
<tr>
<td>GetFieldAsBinary</td>
<td>Read field data as binary data</td>
</tr>
<tr>
<td>GetFieldAsBlob</td>
<td>Read field data as blob</td>
</tr>
<tr>
<td>GetFieldAsBigInt</td>
<td>Read field data as signed 64 bit integer</td>
</tr>
<tr>
<td>GetFieldAsCurrency</td>
<td>Read field data as currency value</td>
</tr>
<tr>
<td>GetFieldAsNumber</td>
<td>Read field data as number (BCD) value</td>
</tr>
</tbody>
</table>

```c
// display all field data on screen
void DisplayData(CTable &ATable, CTRecord &ARecord)
{
    NINT count = ATable.GetFieldCount();
    NINT i;
    CTString str;

    for (i = 0; i < count; i++)
    {
        ARecord.GetFieldAsString(i, str);
        printf("Field %d: %s\n", i, str.c_str());
    }
}
```

The following functions should be used to write fields into the data record buffer:

<table>
<thead>
<tr>
<th>Function</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetFieldAsBool</td>
<td>update field data as Boolean value</td>
</tr>
<tr>
<td>SetFieldAsSigned</td>
<td>update field data as signed integer value</td>
</tr>
<tr>
<td>SetFieldAsUnsigned</td>
<td>update field data as unsigned integer value</td>
</tr>
<tr>
<td>SetFieldAsDate</td>
<td>update field data as date value</td>
</tr>
<tr>
<td>SetFieldAsTime</td>
<td>update field data as time value</td>
</tr>
</tbody>
</table>
### Function | Explanation
--- | ---
SetFieldAsMoney | update field data as money value
SetFieldAsFloat | update field data as double value
SetFieldAsDateTime | update field data as time stamp value
SetFieldAsString | update field data as string value
SetFieldAsBinary | update field data as binary data
SetFieldAsBlob | update field data as blob
SetFieldAsBigInt | update field data as signed 64 bit integer
SetFieldAsCurrency | update field data as currency value
SetFieldAsNumber | Read field data as number (BCD) value

```cpp
// add new record
void AddRecord(CTRecord &ARecord, const CTString& name, const CTString& address, const CTString& phone)
{
    // clear the record buffer
    ARecord.Clear();

    // populate the record buffer with field data
    ARecord.SetFieldAsString(0, name);
    ARecord.SetFieldAsString(1, address);
    ARecord.SetFieldAsString(2, phone);

    // add the new record
    ARecord.Write();
}
```

The `SetFieldAs...()` methods will also clear the null bit flag for the updated field.

When you invoke one of the `GetFieldAs...()` or `SetFieldAs...()` methods, you pass the field number or field name and the data you want to read from or write to the data record buffer.

If the type of the field you are trying to read from, or write to, is different to the type of the data specified by the `GetFieldAs...()` or `SetFieldAs...()` methods, the record manager will automatically convert the field data type to match the data type of the parameter of passed by one of the `GetFieldAs...()` or `SetFieldAs...()` method.

For example, if you are writing a report generator application that displays the fields of a record on screen, you can read all fields in the record buffer with `GetFieldAsString()` and the record manager will convert the different field types to string. Boolean field type is converted as "True" or "False", numeric values are converted to string and dates and times use the session wide default date and time formats.

#### Automatic data type conversion

Some automatic data type conversions may not be possible, and an error (`CTDBRET_CANTCONVERT` or `CTDBRET_INVTYPE`) will occur. This will happen, for instance, if the method `CTRecord::SetFieldAsDate()` is used to store data in a field whose type is `CT_BOOL`.
The table below presents the valid automatic conversions for the fields and the `GetFieldAs...()` or `SetFieldAs...()` methods.

<table>
<thead>
<tr>
<th>Field Type*1</th>
<th>Function ...SetFieldAs... or ...GetFieldAs...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S i g n e d</td>
</tr>
<tr>
<td><code>CT_BOOL</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_TINYINT</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_UTINYINT</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_SMALLINT</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_USMALLINT</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_INTEGER</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_UINTEGER</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_MONEY</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_DATE</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_TIME</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_FLOAT</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_DOUBLE</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_TIMESTAMP</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_EFLOAT</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_BINARY</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_CHARS</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_FPSTRING</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_F2STRING</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_F4STRING</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_BIGINT</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_NUMBER</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_CURRENCY</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_PSTRING</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_VARBINARY</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_LVB</code></td>
<td>X</td>
</tr>
<tr>
<td><code>CT_VARCHAR</code></td>
<td>X</td>
</tr>
</tbody>
</table>

*1 Field Type: Signed, Unsigned, Big Integer, Number, Float, Money, Currency, Date, Time, DateTime, String, Binary, Blob.
*1 - These are the c-treeDB field types. The old c-tree Plus field types are also valid and may be used. For new projects, the best approach is to use the new field type convention presented in this table, since this is the naming convention for the c-treeACE SQL field types.

*2 - The value being set must be equal to either “TRUE” or “FALSE” (case insensitive), otherwise the function will fail with CTDBRET_CANTCONVERT (4042).

For every row in the table above, there is a highlighted column. This column represents the natural function that should be used to update that particular field. As discussed, though, any other function (column) marked with an “X” can be used, and the appropriate conversions will be performed.

**Null field support**

The c-treeDB record manager API implements the underlying support for null fields, but does not enforce it. If a table was created with null field support, the record manager will set or clear the null bit flag for fields, but there is no check to see if null fields are allowed or not.

When CTRecord::Clear() clears a record buffer, all fields are marked as null fields. As you update the field contents by calling SetFieldAs...() methods, the null field for the respective field is cleared indicating that the field contains data.

A specific field can be cleared, and its null field flag set, by calling CTRecord::ClearField() method. Call CTRecord::IsNullField() to check if a field null flag is set.

```c
// if a field 0 is not null, clear the field
if (!ARecord.IsNullField(0))
{
    ARecord.ClearField(0);
}
```

The c-treeDB null flag controls the NOT NULL property of a column. Setting this to column has no effect on individual record null values: it is NOT enforced at the c-treeDB layer. This attribute only applies to the c-treeACE SQL layer for constraint on values. It is useful to set this flag in c-tree data files before SQL import such that the property is maintained.

**Field defined size**

Call CTRecord::GetFieldSize() to retrieve the field defined size. The defined field size is the size of the field declared at the time the table was created.

**Field actual length**

Call CTRecord::GetFieldLength() to retrieve the field actual size. Variable length fields such as CT_VARCHAR and CT_VARBINARY actual size in a record may vary from the defined size.

```c
// check if field 0 actual size is different from defined size
if (ARecord.GetFieldSize(0) != ARecord.GetFieldLength(0))
{
    printf("Field 0 defined and actual sizes are different\n");
}
```
Field address in record buffer

The field address in a record buffer will also vary if the field is located in the variable portion of a record. Call `ctdbGetFieldAddress()` to retrieve the field address in the record buffer. Fields in the fixed portion of a record buffer will always return the same address.

`CTRecord::GetFieldAddress()` takes the field number as a parameter and returns a pointer to the address of the field in the record buffer. `CTRecord::GetFieldAddress()` returns NULL if an error occurs and the field address cannot be retrieved.

Field offset in record buffer

A field offset address in the record buffer can also be retrieved by calling `CTRecord::GetFieldOffset()`. The offsets of fields in the fixed portion of the record never change, while the offsets of fields in the variable portion of a record may change to accommodate the different sizes of variable length fields.

Check if field is fixed or variable length

Call `CTRecord::IsVariableField()` method to check if a field is variable length or not. `CTRecord::IsVariableField()` return YES if the field is variable length.

```c++
// check if field 0 is variable length
if (ARecord.IsVariableField(0))
{
    printf("Field 0 is variable length\n");
}
else
{
    printf("Field 0 is fixed length\n");
}
```

Using field names

Most `CTRecord` class methods that operate on fields have overloaded methods that will accept a field number or a field name as a parameter to indicate which field to operate on.

Clearing the record buffer

Call `CTRecord::Clear()` to clear the record buffer. The record buffer is cleared, the full bit flag for all fields are cleared, the new record flag is set to false, the edited record flag is cleared, and the record offset is set to zero.

```c++
// clear the record buffer
CTRecord ARecord(ATable);
try
{
    ARecord.Clear();
}
catch (CTException &err)
{
    printf("Record clear failed with error %d\n", err.GetErrorCode());
}
```
Clearing a field
A field can be cleared, and its null field flag set, by calling `CTRecord::ClearField()` method. Call `CTRecord::IsNullField()` to check if a field null flag is set.

```cpp
// if a field 0 is not null, clear the field
if (!ARecord.IsNullField(0))
{
    try
    {
        ARecord.ClearField(0);
    }
    catch (CTException &err)
    {
        printf("Clear field failed with error %d\n", err.GetErrorCode());
    }
}
```

Adding new records
To add a new record to a table, you need to perform the following actions:

1. Clear the record buffer
2. Populate the fields by calling the methods `CTRecord::SetFieldAs...()`
3. Add the record by calling `CTRecord::Write()`

```cpp
// clear the record buffer
ARecord.Clear();

// populate the record buffer
ARecord.SetFieldAsSigned(0, 1000);
ARecord.SetFieldAsString(1, "Joe Smith");

// write the record
try
{
    ARecord.Write();
}
catch (CTException &err)
{
    printf("Add record failed with error %d\n", err.GetErrorCode());
}
```

A record is added to a table if the new record flag and edited flag are both set. `CTRecord::Clear()` sets the new record flag, while one of the `CTRecord::SetFieldAs...()` functions will set the edited record flag.

If you clear a record and, without populating the record buffer with data, a call to `CTRecord::Write()` returns no error code but no data record is written to disk, since the new record flag is set but the edited record flag is cleared.

If you wish to write a cleared or blank record to disk, you must clear the record buffer, set the edited record flag by calling `ctdbSetEditedRecord()` and then write the record by calling `CTRecord::Write()`.

```cpp
// clear the record buffer
ARecord.Clear();

// set the edited record flag
ARecord.SetEdited(YES);
```
// write the record
try
{
    ARecord.Write();
} catch (CTException &err)
{
    printf("Write record failed with error %d\n", err.GetErrorCode());
}

**Updating existing records**

To update an existing record, you need to perform the following actions:

1. Read the record from disk by calling one of the `CTRecord::First()`, `CTRecord::Last()`, `CTRecord::Next()`, `CTRecord::Prev()`, `CTRecord::SeekRecord()`, `CTRecord::Find()` or `CTRecord::FindTarget()` methods.
2. Update one or more fields with one of the `CTRecord::SetFieldAs...()` methods.
3. Write the record by calling `CTRecord::Write()`.

```cpp
// update the first record
ARecord.First();

// change the first field
ARecord.SetFieldAsString(0, "Joe Smith");
// write the record
try
{
    ARecord.Write();
} catch (CTException &err)
{
    printf("Update record failed with error %d\n", err.GetErrorCode());
}
```

A record is updated when the new record flag is cleared and the edited record flag is set. The edited record flag is set when a record is read from disk and it indicates that this is an existing record.

If you read a record from disk, and without updating the record data, call `CTRecord::Write()`, no data record is written to disk, since the edited record flag is cleared.

A record update can be forced by setting the record edited flag with a call to `CTRecord::SetEdited()` method.

```cpp
// force update of first record
ARecord.First();

// set the edited flag
ARecord.SetEdited(YES);

// write the record
try
{
    ARecord.Write();
} catch (CTException &err)
{
    printf("Update record failed with error %d\n", err.GetErrorCode());
}
```
A record can be duplicated by reading it from disk, setting the edited record flag by calling `CTRecord::SetEdited()` and setting the new record flag by calling `CTRecord::SetNew()`, and calling `CTRecord::Write()` to write it to disk. Please note that a record can only be duplicated if the table has no indices with unique keys.

```cpp
// duplicate the first record
ARecord.First();

// set the edited flag
ARecord.SetEdited(YES);

// set the new record flag
ARecord.SetNew(YES);

// write the record
try
{
    ARecord.Write();
} catch (CTException &err)
{
    printf("Write record failed with error %d\n", err.GetErrorCode());
}
```

### Deleting records

To delete a record, you need to perform the following actions:

1. Read the record from disk by calling one of the `CTRecord::First()`, `CTRecord::Last()`, `CTRecord::Next()`, `CTRecord::Prev()`, `CTRecord::SeekRecord()`, `CTRecord::Find()` or `CTRecord::FindTarget()` methods.

2. Delete the record by calling `CTRecord::Delete();`

```cpp
// delete the first record
ARecord.First();

// delete the record
try
{
    ARecord.Delete();
} catch (CTException &err)
{
    printf("Delete record failed with error %d\n", err.GetErrorCode());
}
```

### Record properties

#### New record flag

The new record flag indicates if a record has been cleared and not written to disk yet. The record manager uses this flag to decide if a record should be added or updated.

The new record flag is set when the record handle is allocated or when `CTRecord::Clear()` clears the record. The new record flag is cleared when a record is read from disk by calling one of the `CTRecord::First()`, `CTRecord::Last()`, `CTRecord::Next()`, `CTRecord::Prev()`, `CTRecord::SeekRecord()`, `CTRecord::Find()` or `CTRecord::FindTarget()` methods.
Use `CTRecord::SetNew()` to set the record new flag or `CTRecord::IsNew()` to check if a record is new or not.

**Edited record flag**

A record is only written to disk if the edited record flag is set. This flag is set when the record buffer is modified with a call to one of the `CTRecord::SetFieldAs...()` methods or by calling `CTRecord::SetEdited()` method. Use `CTRecord::IsEdited()` to check if a record has been edited or not.

**Record offset**

This property holds the current record offset of a record. If a record is cleared, the record offset property is zero. All records in a table, even the first record, will have a record offset value greater than zero.

You can retrieve the record offset value by calling `CTRecord::GetRecordPos()` method. You can set the record offset property, and load the record data at the offset, by calling `CTRecord::SeekRecord()`.

**Record count**

Use `CTRecord::GetRecordCount()` to retrieve the total number of records in a table. This is a read only property.

```cpp
// check if table is empty
if (ARecord.GetRecordCount() > 0)
{
    printf("Table is not empty\n");
}
```

**Record ROWID**

Use `CTRecord::GetRowid()` method to retrieve the ROWID value for the current record. This is a read only property.

```cpp
// retrieve the first record rowid
CTROWID rowid = ARecord.GetRowid();
```

**Record Locking**

A record lock can be acquired by performing the following actions:

1. Read a record from disk by calling one of the `CTRecord::First()`, `CTRecord::Last()`, `CTRecord::Next()`, `CTRecord::Prev()`, `CTRecord::SeekRecord()`, `CTRecord::Find()`, or `CTRecord::FindTarget()` methods.
2. Lock the record by calling `CTRecord::LockRecord()`.

Session wide locks are better suited to implement data integrity because the records are locked as they are read from disk. Since a record lock can only be applied after the record is read, there is a window of opportunity for the record be modified or deleted by another thread before the record lock is acquired.

Release a record lock by calling `CTRecord::UnlockRecord()` or by calling `CTRecord::LockRecord()` with `CTLOCK_FREE` mode.
**Check if a record is locked**

Call `CTRecord::GetRecordLock()` to retrieve the current lock mode acquired for the current record. If the record is not locked, `CTRecord::GetRecordLock()` returns `CTLOCK_FREE`. If the current record is locked, `CTRecord::GetRecordLock()` returns the record lock mode.

```cpp
// check if record is locked
if (ARecord.GetRecordLock() != CTLOCK_FREE)
{
    printf("The record is locked\n");
}
```

The following record modes are returned by `ctdbGetRecordLock()`:

<table>
<thead>
<tr>
<th>Lock mode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CTLOCK_FREE</code></td>
<td>The record is not locked</td>
</tr>
<tr>
<td><code>CTLOCK_READ</code></td>
<td>The record has a read lock</td>
</tr>
<tr>
<td><code>CTLOCK_WRITE</code></td>
<td>The record has a write lock</td>
</tr>
</tbody>
</table>

**Demoting record locks**

If a record has acquired a write lock, it is possible to change or demote the write lock to a read lock, if the record has not been updated. Use the `CTRecord::LockRecord()` method to demote a record with a write lock to a read lock.

```cpp
// demote a write lock to a read lock
if (ARecord::GetRecordLock() == CTLOCK_WRITE)
{
    try
    {
        ARecord.LockRecord(CTLOCK_READ);
    }
    catch (CTException &err)
    {
        printf("Demote lock failed with error %d\n", err.GetErrorCode());
    }
}
```
3.6 Data Types

The topics below list the data types supported by c-treeDB.

Scalar Types

These are the data types that map directly to C or C++ scalar types. c-treeDB allows the use of c-tree scalar data types such as:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>Declared as 8-bit char. This is the equivalent of C type char.</td>
</tr>
<tr>
<td>UTEXT</td>
<td>Declared as 8-bit unsigned char. This is the equivalent of C type unsigned char.</td>
</tr>
<tr>
<td>COUNT</td>
<td>Declared as 16-bit integer. This is the equivalent of C type short.</td>
</tr>
<tr>
<td>UCOUNT</td>
<td>Declared as 16-bit unsigned integer. This is the equivalent of C type unsigned short.</td>
</tr>
<tr>
<td>NINT</td>
<td>Declared as the system natural integer. This is equivalent of C type integer.</td>
</tr>
<tr>
<td>UINT</td>
<td>Declared as the system natural unsigned integer. This is the equivalent of C type unsigned integer.</td>
</tr>
<tr>
<td>LONG</td>
<td>Declared as 32-bit signed integer. This is the equivalent of C type long for 32-bit CPUs.</td>
</tr>
<tr>
<td>ULONG</td>
<td>Declared as 32-bit unsigned integer. This is the equivalent of C type unsigned long for 32-bit CPUs.</td>
</tr>
</tbody>
</table>

c-treeDB also exposes the following data types, based on c-tree scalar types. These c-treeDB types also provide a one-to-one relationship with c-treeDB field types:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTBOOL</td>
<td>Declared as an integer, this type holds Boolean values using the predefined Boolean constants YES for true and NO for false.</td>
</tr>
<tr>
<td>CTSIGNED</td>
<td>Declared as a signed long (32-bit signed integer), this type holds any scalar value: 8-bit signed char (TEXT), 16-bit signed integer (COUNT), integers (NINT), and 32-bit signed integers (LONG).</td>
</tr>
<tr>
<td>CTUNSIGNED</td>
<td>Declared as an unsigned long (32-bit unsigned integer), this type holds any scalar value: 8-bit unsigned char (UTEXT), 16-bit unsigned integer (UCOUNT), unsigned integers (UINT), and 32-bit unsigned integers (ULONG).</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>Declared as C type double.</td>
</tr>
<tr>
<td>CTFLOAT</td>
<td>Declared as C type double.</td>
</tr>
</tbody>
</table>

All the types in this category can be manipulated using the normal operators defined for the C and C++ language.

Date Types

CTDATE is declared as an unsigned 32-bit integer. The date is stored as the number of days since March 1st 1700. The CTDATETYPE was implemented to be compatible with FairCom’s
r-tree and ODBC products. Hence, if a CTDATE type has a value of 1, it means that the date is March 1st 1700, while a value of 2 means March 2nd 1700 and so on.

Since CTDATE is a 32-bit unsigned integer, numerical operations with dates can be done using standard C operators. For example, to add 10 days to a date use the normal C operation date + 10 (assuming that date is a CTDATE type). To get the difference in days of 2 dates, just subtract one date from another.

c-treeDB provides methods and functions to allow the user to manipulate the CTDATE type in their applications.

<table>
<thead>
<tr>
<th>Method</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pack</td>
<td>Pack a date as year, month and day and store the result in pDate. The date is checked and ctdbDatePack() returns CTDBRET_OK if the date was packed correctly.</td>
</tr>
<tr>
<td>Unpack</td>
<td>Update a CTDATE value into year, month and day.</td>
</tr>
<tr>
<td>ToString</td>
<td>Convert a packed CTDATE value into string. The format of the date in string form is given by DateType().</td>
</tr>
<tr>
<td>StringToDate</td>
<td>Convert a date in string format to a packed CTDATE value. The format of the date in string form is given by DateType() as described above.</td>
</tr>
<tr>
<td>Day</td>
<td>Retrieve the day component of a packed date type.</td>
</tr>
<tr>
<td>Month</td>
<td>Retrieve the month component of a packed date type.</td>
</tr>
<tr>
<td>Year</td>
<td>Retrieve the year component of a packed date type.</td>
</tr>
<tr>
<td>IsLeapYear</td>
<td>Indicate if the year component of the packed date type is a leap year.</td>
</tr>
<tr>
<td>DayOfWeek</td>
<td>Retrieve the day of the week for a given packed date type. Sunday is 0, Monday is 1, Tuesday is 2, and so on.</td>
</tr>
<tr>
<td>CurrentDate</td>
<td>Retrieve the current system date and store it in CTDATE packed format.</td>
</tr>
</tbody>
</table>

The possible date formats for string conversion are:

<table>
<thead>
<tr>
<th>c-treeDB Symbolic Constant</th>
<th>c-treeDB .NET Symbolic Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTDATE_MDCY</td>
<td>MDCY_DATE</td>
<td>Date is mm/dd/ccyy</td>
</tr>
<tr>
<td>CTDATE_MDY</td>
<td>MDY_DATE</td>
<td>Date is mm/dd/yy</td>
</tr>
<tr>
<td>CTDATE_DMCY</td>
<td>DMCY_DATE</td>
<td>Date is dd/mm/ccyy</td>
</tr>
<tr>
<td>CTDATE_DMY</td>
<td>DMY_DATE</td>
<td>Date is dd/mm/yy</td>
</tr>
<tr>
<td>CTDATE_CYMD</td>
<td>CYMD_DATE</td>
<td>Date is ccymmdd</td>
</tr>
<tr>
<td>CTDATE_YMD</td>
<td>YMD_DATE</td>
<td>Date is yymmd</td>
</tr>
</tbody>
</table>
**Time Types**

`CTTIME` is declared as a 32-bit unsigned integer, stored as the number of seconds since midnight. The `CTTIME` type was implemented to be compatible with FairCom’s r-tree and ODBC products.

c-treeDB provides functions and methods to allow the user to manipulate the `CTTIME` type in their applications.

<table>
<thead>
<tr>
<th>Method</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pack</td>
<td>Pack a time from hour, minute, and second into a <code>CTTIME</code> form.</td>
</tr>
<tr>
<td>Unpack</td>
<td>Unpack a <code>CTTIME</code> time into hour, minute, and second.</td>
</tr>
<tr>
<td>TimeToString</td>
<td>Convert a packed time value into string format.</td>
</tr>
<tr>
<td>StringToTime</td>
<td>Convert a time in string form into a packed time. <code>TimeType()</code> gives the string form of the time value.</td>
</tr>
<tr>
<td>Hour</td>
<td>Retrieve the hour component of a packed time. The hour return is a 24-hour value. Example 11 pm is return as 23.</td>
</tr>
<tr>
<td>Minute</td>
<td>Retrieve the minute component of a packed time.</td>
</tr>
<tr>
<td>Second</td>
<td>Retrieve the second component of a packed time.</td>
</tr>
<tr>
<td>CurrentTime</td>
<td>Retrieve the current system time.</td>
</tr>
</tbody>
</table>

Time Types can be one of the following string time formats:

<table>
<thead>
<tr>
<th>c-treeDB Symbolic Constant</th>
<th>c-treeDB .NET Symbolic Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CTTIME_HMSP</code></td>
<td><code>HMSP_TIME</code></td>
<td>Time is hh:mm:ss am</td>
</tr>
<tr>
<td><code>CTTIME_HMP</code></td>
<td><code>HMP_TIME</code></td>
<td>Time is hh:mm am</td>
</tr>
<tr>
<td><code>CTTIME_HMS</code></td>
<td><code>HMS_TIME</code></td>
<td>Time is hh:mm:ss (24 hour)</td>
</tr>
<tr>
<td><code>CTTIME_HM</code></td>
<td><code>HM_TIME</code></td>
<td>Time is hh:mm (24 hour)</td>
</tr>
<tr>
<td><code>CTTIME_MIL</code></td>
<td><code>MIL_TIME</code></td>
<td>Time is hhmm (military)</td>
</tr>
<tr>
<td><code>CTTIME_HHMST</code></td>
<td></td>
<td>Time is hh:mm:ss.ttt (24 hour)</td>
</tr>
</tbody>
</table>

**Date/Time (Timestamp) Types**

`CTDATETIME` is declared as a double type and stores both the date and time. This type is also called a time stamp. The date component of a `CTDATETIME` value is stored as the value on the left of the decimal point, while the time component is stored as the value on the right side of the decimal point.

c-treeDB provides functions and methods to manipulate the `CTDATETIME` type.
### Numeric Types

Numeric types are used to manipulate numeric values that are too large for the scalar types or numeric values representing currency values. c-treeDB implements the following numeric types:

**CTBIGINT**

*CTBIGINT* is a 64-bit signed integer type. Today most C compilers are capable of dealing with 64-bit integers. In the Windows operating system, Borland, Microsoft and Watcom use a `__int64` type to represent native 64-bit integers, while in other operating systems such as Unix and Linux, 64-bit integers are represented as long long types.

The following set of functions converts of *CTBIGINT* into other c-treeDB types.

<table>
<thead>
<tr>
<th>Method</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AsLong</strong></td>
<td>Convert a <em>CTBIGINT</em> value to a long (32-bit signed integer). If the <em>CTBIGINT</em> value is too large for the conversion, <strong>AsLong()</strong> returns <em>CTDBRET_OVERFLOW</em> or <em>CTDBRET_UNDERFLOW</em> errors.</td>
</tr>
<tr>
<td><strong>AsFloat</strong></td>
<td>Convert a <em>CTBIGINT</em> value to <em>CTFLOAT</em>.</td>
</tr>
<tr>
<td><strong>AsString</strong></td>
<td>Convert a <em>CTBIGINT</em> value to string.</td>
</tr>
<tr>
<td><strong>CTBigint(CTSTRring)</strong></td>
<td>Convert a string to <em>CTBIGINT</em> value.</td>
</tr>
</tbody>
</table>

**CTMONEY**

*CTMONEY* represents a currency value in a 32-bit signed integer. The last two decimal digits of the value are used as the decimal part of the value. For example, a currency value of 123.45 is represented with *CTMONEY* as 12345. A currency value of 1 is represented in *CTMONEY* as 100.
Since all the operations performed on CTMONEY values are integer operations, this type offers exact currency value capabilities that do not need large values or large precision at excellent performance that is very close to 32-bit integer performance.

C-treeDB provides the following set of functions to manipulate CTMONEY values, including functions for performing the basic arithmetic operations on CTMONEY values. These set of basic arithmetic operations are especially important in multiplication and division of CTMONEY values. Take for example the multiplication of two values such as 123.45 and 67.89. These values are represented in a CTMONEY type as 12345 and 6789 respectively. An integer multiplication on the values above, would give the result 83810205, which is 838102.05 in CTMONEY representation. This result is clearly wrong since the expected result would be 8381.02. The same principle applies to division operation. By using the CTMONEY API provided by C-treeDB, the user will be able to operate correctly with CTMONEY values.

Note: Conversions to CTMoney in C++ use a constructor in the format: CTMoney(const [CT type]& value) and are represented below as CTMoney([CT type]). Also, C++ overrides operators (such as +, -, *, and /) so that the result returned is the same type as the items operated upon. For example, the following command would put the result in c as type CTMoney.

```cpp
C = (CTMoney) a + (CTMoney) b;
```

In C++, comparison operators can be used for any CT types; the return is a Boolean indicating true or false.

<table>
<thead>
<tr>
<th>Method</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsLong</td>
<td>Convert a CTMONEY value to LONG. Only the integer portion of the CTMONEY value is converted to LONG, as the decimal portion of the CTMONEY value is ignored for the conversion.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>Convert CTMONEY value to CTFLOAT.</td>
</tr>
<tr>
<td>AsString</td>
<td>Convert CTMONEY value to string.</td>
</tr>
</tbody>
</table>

CTCURRENCY

CTCURRENCY represents a currency value in a 64-bit signed integer. The last four decimal digits of the value are used as the decimal part of the value. Example: a currency value of 123.45 is represented with CTCURRENCY type as 1234500. A currency value of 1 is represented in CTCURRENCY as 10000.

Since all the operations performed on CTCURRENCY values are integer operations, this type offers exact currency value capabilities with large value capabilities and good precision at excellent performance that is very close to 64-bit integer performance.

pResult provides the following set of functions to manipulate CTCURRENCY values, including functions for performing the basic arithmetic operations on CTCURRENCY values. This set of basic arithmetic operations are especially important in multiplication and division of CTCURRENCY values. Take for example the multiplication of two values such as 123.45 and 67.89. These values are represented in a CTCURRENCY type as 1234500 and 678900 respectively. An integer multiplication on the values above would yield 838102050000, which is 83810205.0000 in CTCURRENCY representation. This result is clearly wrong since the expected result would be 8381.02. The same principle applies to division operation. By using the
CTCURRENCY API provided by pResult, the user will be able to operate correctly with CTCURRENCY values.

Note: Conversions to CTCurrency in C++ use a constructor in the format: CTCurrency(const [CT type]& value) and are represented below as CTCurrency([CT type]). Also, C++ overrides operators (such as +, -, *, and /) so that the result returned is the same type as the items operated upon. For example, the following command puts the result in c as type CTMoney.

\[ c = (\text{CTCurrency}) \ a + (\text{CTCurrency}) \ b; \]

In C++, comparison operators can be used for any CT types; the return is a Boolean indicating true or false:

<table>
<thead>
<tr>
<th>Method</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTCurrency(CTMoney)</td>
<td>Convert a CTMONEY value to CTCURRENCY value.</td>
</tr>
<tr>
<td>AsMoney</td>
<td>Convert a CTCURRENCY value to CTMONEY value.</td>
</tr>
<tr>
<td>AsLong</td>
<td>Convert a CTCURRENCY value to a LONG value.</td>
</tr>
<tr>
<td>AsBigint</td>
<td>Convert a CTCURRENCY value to CTBIGINT value.</td>
</tr>
<tr>
<td>CTCurrency(CTBigint)</td>
<td>Convert a CTBIGINT value to CTCURRENCY value.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>Convert a CTCURRENCY value to a CTFLOAT value.</td>
</tr>
<tr>
<td>AsString</td>
<td>Convert a CTCURRENCY value to a string.</td>
</tr>
<tr>
<td>StringToCurrency</td>
<td>Convert a string to a CTCURRENCY value.</td>
</tr>
</tbody>
</table>

CTNUMBER

CTNUMBER corresponds to a number with a given precision (maximum number of digits) and scale (the number of digits to the right of the decimal point). Numeric values can have maximum values of 32 digits precision and scale of 0 (this is 9999999999999999999999999999999999), which can represent very large exact values.

pResult provides the following set of functions to manipulate this very powerful number representation.

Note: Conversions to CTNumber in C++ use a constructor in the format: CTNumber(const [CT type]& value) and are represented below as CTNumber([CT type]). Also, C++ overrides operators (such as +, -, *, and /) so that the result returned is the same type as the items operated upon. For example, the following command would put the result in c as type CTMoney.

\[ c = (\text{CTNumber}) \ a + (\text{CTNumber}) \ b; \]

In C++, comparison operators can be used for any CT types; the return is a Boolean indicating true or false:

<table>
<thead>
<tr>
<th>Method</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsMoney</td>
<td>Convert CTNUMBER to CTMONEY.</td>
</tr>
<tr>
<td>AsLong</td>
<td>Convert CTNUMBER to LONG.</td>
</tr>
<tr>
<td>AsBigint</td>
<td>Convert CTNUMBER to CTBIGINT.</td>
</tr>
<tr>
<td>Method</td>
<td>Operation</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AsCurrency</td>
<td>Convert <code>CTNUMBER</code> to <code>CTCURRENCY</code>.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>Convert <code>CTNUMBER</code> to <code>CTFLOAT</code>.</td>
</tr>
<tr>
<td>AsString</td>
<td>Convert <code>CTNUMBER</code> to string.</td>
</tr>
<tr>
<td>Zero</td>
<td>Set the value pointed by <code>pNumber</code> to zero.</td>
</tr>
<tr>
<td>IsZero</td>
<td>Return YES if the value is zero.</td>
</tr>
<tr>
<td>Round</td>
<td>Round the value pointer by <code>num</code> to the number of decimal digits (digits to the right of the decimal point) indicated by <code>scale</code>.</td>
</tr>
<tr>
<td>DecimalDigits</td>
<td>Given the <code>CTNUMBER</code> data, stores the number of digits to the left of the decimal point in <code>digit_before</code> and stores the number of digits to the right of decimal point in <code>digit_after</code>.</td>
</tr>
</tbody>
</table>
3.7 Data Integrity

This section describes the two key elements of ensuring data integrity, record locking and transaction control.

Transactions

There are two major aspects to transaction processing, atomicity and automatic recovery. These are related yet different aspects of transaction processing, and not all products supply both. c-treeDB provides a set of functions and file modes that cover both aspects of transaction processing.

Atomicity

Often, when updating a table, you perform several functions in a group. For instance, when creating an invoice, you update several tables: the account balance in the customer file, the invoice file, an invoice detail file, inventory records, and others. It is important that all of these actions take place to keep the files synchronized. If some of the actions take place, but not all, your files may be out of sync, and it can be difficult to correct the problem later. If one action cannot take place, it would be best to not let any take place. We call this atomicity. The c-treeDB API provides functions that provide this feature. You can mark a set of operations so that none will take place unless they can all take place. The API goes beyond this, allowing you to create "savepoints" where you can partially back out a group of operations, and "roll back" transactions to a given point, so that you can restore your data back to a state that it was in sometime in the past.

Automatic Recovery

Once you establish full transaction processing by creating tables using the `CTCREATE_TRNLOG` mode, you can take advantage of the automatic recovery feature. Atomicity will generally prevent problems of files being partially updated. However, there are still situations where a system crash can cause data to be lost. Once you have signaled the end of a transaction, there is still a "window of vulnerability" while the application is actually committing the transaction updates to disk. In addition, for speed considerations some systems buffer the data files and indices, so that updates may not be flushed to disk immediately. If the system crashes, and one of these problems exists, the recovery logic detects it. If you set up the system for automatic file recovery, the recovery logic automatically resets the table back to the last, most complete, state that it can. If any transaction sets have not been completed, or "committed", they will not affect the table.

Error During Automatic Recovery

An error 14, `FCRP_ERR`, indicates that c-treeACE Server detected that files appear corrupt at open. This occurs if files have been updated but not properly closed. They were not processed by automatic recovery so they are in an unknown (inconsistent) state.

If your transaction logs are corrupted, preventing automatic recovery from occurring, you can either:
• Restore from a backup and reapply changes if available (e.g., from application log or forward roll of good transaction logs you have saved).

or

• Rebuild the files, which will clear the error 14 but will still leave the files in a possibly inconsistent state. In this situation the files will not be guaranteed to be consistent as of any point in time; they can contain a mixture of old/new data, and the data files may not match the index files, due to caching.

Creating tables for transaction processing

Only tables created with the modes `CTCREATE_PREIMG` and `CTCREATE_TRNLOG` will participate in a transaction.

Tables created with `CTCREATE_PREIMG` mode will participate in a transaction and only transaction atomicity is applied to them.

Tables created with `CTCREATE_TRNLOG` have all the attributes of `CTCREATE_PREIMG` but will also generate the transaction logs necessary for automatic recovery.

Starting a transaction

Using our example from above, you don’t want to have the transaction group involve more than one invoice. You also don’t want it to involve less than a whole invoice.

Record locks are held on updated records for the duration of the transaction, so you don’t want to make the transaction group too large or it will consume the system resources. On the other hand, you may not want to make the transaction group too small or the effect of grouping actions is lost.

The Begin method of classes `CTSession()`, `CTDatabase()`, `CTTable()` and `CTRecord()` starts a new transaction. Choose logical groups of file updates to be delimited as transactions.

```c++
// start a new transaction
try
{
    ARecord.Begin()
}
catch (CTException &err)
{
    printf("Begin transaction failed with error %d\n", err.GetErrorCode());
}
```

Terminating a transaction

When all update operations have been completed, terminate a transaction by calling the Commit method to commit all changes.

```c++
// Start transaction
ARecord.Begin();
// write the record
try
{
    ARecord.Write();
    ARecord.Commit();
}
catch (CTException &err)
```
Begin
{ 
  ARecord.Abort();
  printf("Commit transaction failed with error %d\n", err.GetErrorCode());
}

Call **Abort()** to terminate the transaction and abort all changes since the start of the transaction.

---

### Save Points

There are times when you want to abort only a portion of a transaction. You may be processing several optional paths of a program, going down one branch, then backing out and trying another branch. It may be possible that you don’t want any of the updates to occur until you are completely finished, but you want the flexibility to back out part of the updates. Another possibility would be if you have run into some form of update error, such as an add record failing due to a duplicate key. You would want to back up to a prior point, correct the problem, and continue. The c-tree Servers let you implement this by using savepoints.

A savepoint is a temporary spot in the transaction that you may want to rollback to without having to abort the entire transaction. During a transaction, when you want to put a placeholder in the process, call the **SetSavePoint()** method. This does not commit any of the updates. The function returns a savepoint number, which you should keep track of. You can make as many SavePoint calls as you wish during a transaction, and each time you will be given a unique savepoint number.

When you decide that you want to rollback to a savepoint previously saved by a call to **SetSavePoint()**, issue a **RestoreSavePoint()** call. You should pass to RestoreSavePoint() the savepoint number that you saved. This returns your data to the state it was at the point you issued the specified SetSavePoint() call, without aborting the entire transaction.

---

### Locking

The most significant difference between coding applications for single-user environments and multi-user environments (including local area networks) has to do with performing record updates (rewrites) and record deletions.

The basic problem is that to perform a record update or delete in a multi-user system, you must own a record lock on the record of interest. Locks should be acquired when the user reads the record in preparation for updates, and should not relinquish the lock until the update is completed. However, one must be careful to help ensure that locks are held for the shortest time possible or the performance of the application will be adversely affected.

---

### Starting locks

Start acquiring locks by passing the appropriate lock mode to the Lock method inherited by **CTSession()**, **CTDatabase()**, **CTTable()**, and **CTRecord()** from **CTBase()**.

```c
// start locking
ARecord.Lock(CTLOCK_WRITE_BLOCK);
```

After a successful call to Lock, the c-treeDB API locks all records as they are read using the lock mode passed to Lock. Suspend record locking temporarily by calling the Lock function with the mode **CTLOCK_SUSPEND**. Suspending locks does not release any locks, but while locks are suspended, no record reads are automatically locked.
// suspend locking
ARecord.Lock(CTLOCK_SUSPEND);

Suspended locking can be resumed by calling Lock with a resume lock mode:
CTLOCK_RESUME_READ, CTLOCK_RESUME_LOCK_BLOCK, CTLOCK_RESUME_WRITE,
or CTLOCK_RESUME_WRITE_BLOCK.

// resume locking
try{
    ARecord.Lock(CTLOCK_RESUME_WRITE_BLOCK);
} catch (CTException &err) {
    printf("Resume lock failed with error %d\n", err.GetErrorCode());
}

Lock modes

Use the following lock modes when calling Lock():

<table>
<thead>
<tr>
<th>c-treeDB Lock Modes</th>
<th>c-treeDB .NET Lock Modes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTLOCK_FREE</td>
<td>FREE_LOCK</td>
<td>Free all locks. Free the data record lock.</td>
</tr>
<tr>
<td>CTLOCK_READ</td>
<td>READ_LOCK</td>
<td>Non-blocking read locks. If the lock cannot be acquired an error is returned.</td>
</tr>
<tr>
<td>CTLOCK_READ_BLOCK</td>
<td>READ_BLOCK_LOCK</td>
<td>Blocking read lock. The thread will block until the lock can be acquired.</td>
</tr>
<tr>
<td>CTLOCK_WRITE</td>
<td>WRITE_LOCK</td>
<td>Non-blocking write lock. If the lock cannot be acquired an error is returned.</td>
</tr>
<tr>
<td>CTLOCK_WRITE_BLOCK</td>
<td>WRITE_BLOCK_LOCK</td>
<td>Blocking write lock. The thread will block until the lock can be acquired.</td>
</tr>
<tr>
<td>CTLOCK_RESET</td>
<td>RESET_LOCK</td>
<td>Equivalent to calling Lock with CTLOCK_FREE followed by Lock() with CTLOCK_WRITE.</td>
</tr>
<tr>
<td>CTLOCK_SUSPEND</td>
<td>SUSPEND_LOCK</td>
<td>Temporarily suspend locking.</td>
</tr>
<tr>
<td>CTLOCK_RESTORE_READ</td>
<td>RESTORE_READ_LOCK</td>
<td>To be used after a call to Lock with the CTLOCK_SUSPEND mode. This lock mode restores the lock mode as READ.</td>
</tr>
<tr>
<td>CTLOCK_RESTORE_READ_BLOCK</td>
<td>RESTORE_READ_BLOCK_LOCK</td>
<td>To be used after a call to Lock with the CTLOCK_SUSPEND mode. This lock mode restores the lock mode as READ_BLOCK.</td>
</tr>
<tr>
<td>CTLOCK_RESTORE_WRITE</td>
<td>RESTORE_WRITE_LOCK</td>
<td>To be used after a call to Lock with the CTLOCK_SUSPEND mode. This lock mode restores the lock mode as WRITE.</td>
</tr>
<tr>
<td>CTLOCK_RESTORE_WRITE_BLOCK</td>
<td>RESTORE_WRITE_BLOCK_LOCK</td>
<td>To be used after a call to Lock with the CTLOCK_SUSPEND mode. This lock mode restores the lock mode as WRITE_BLOCK.</td>
</tr>
</tbody>
</table>
### c-treeDB Lock Modes

<table>
<thead>
<tr>
<th>c-treeDB Lock Modes</th>
<th>c-treeDB .NET Lock Modes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTLOCK_RESTORE</td>
<td>CTLOCK_RESTORE _PREVIOUS</td>
<td>To be used after a call to Lock with the CTLOCK_SUSPEND mode. This lock mode restores the same lock mode valid before suspending the lock.</td>
</tr>
</tbody>
</table>

c-treeDB .NET Lock Modes are defined in the LOCK_MODE enum.

### Freeing locks

Locks are freed by calling `Unlock()` or by calling `Lock()` with the CTLOCK_FREE mode. If freeing locks inside an active transaction, the locks of updated records will only be actually freed when the transaction terminates.

```c
// free locks
ARecord.Unlock();
```

### Freeing locks associated with a table

You can free only the locks associated with a table by calling the `CTTable::UnlockTable()` method. Only the locks held for records of the table are released, and all other record locks are kept.

```c
// free locks for a table
try
{
 ATable.UnlockTable();
}
catch (CTException &err)
{
   printf("Unlock table failed with error %d\n", err.GetErrorCode());
}
```

If freeing locks associated with a table inside an active transaction, the locks of updated records will only be actually freed when the transaction terminates.

### 3.8 Working with Resources

It can be advantageous at times to attach auxiliary information to a particular table that does not conform to the record structure of that table. For example, features such as versioning or special flags relating to the status of the table. Generally, this information is not repeated in every record of the table. You could create a special record, with a special key value, that you do not process as you do your regular records. Ultimately, however, this forces exceptional handling routines for this special case and can impose a heavy maintenance cost on the life cycle of the application.

c-treeACE provides a unique feature created for exactly this purpose. c-treeACE Resources. Resources are special variable-length records stored within your data file, whether you use fixed or variable length records. A set of c-treeACE API functions provide access to create, update and delete these resource records. These records do not require a key in an index, therefore your program does not access them via routine data handling functions.

Resources provide critical support for many advanced c-treeACE features. FairCom defined resources allow seamless functionality of many of the c-tree Plus ODBC Drivers, r-tree Report...
Engine, Incremental ISAM functionality, conditional index support, c-treeDB and c-treeACE SQL. These resources continue to be important as new technology is added. Resources are added either automatically by some c-treeACE features or manually by the developer. The use of resources requires the RESOURCES define in the c-treeACE library, which is the default.

This section focuses on user defined resources added by you, the developer, and provides important background information on the use of resources.

**Types of Resources**

There are three general types of resources that can be attached to a file.

**User defined resources**

Information that you wish to store in the table, such as a version number, or an infrequently accessed counter. Use resources to store information associated with a table that varies from the type of information stored repetitively in the data records.

**FairCom defined resources**

There is a variety of information that, under certain circumstances, FairCom wishes to store in the table. This can be information relating to IFIL structures, alternate collating sequences, special characters for key padding, and so forth. Usually you do not access this information directly. It is available to a variety of c-treeDB functions that use it.

**Third party resources**

As other developers create utilities integrating with c-treeACE, FairCom assigns resource identifiers when necessary.

**Resource Identification**

Within a given data file, a Resource is identified by three elements.

<table>
<thead>
<tr>
<th>Resource Element</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>unsigned long integer</td>
</tr>
<tr>
<td>Number</td>
<td>unsigned long integer</td>
</tr>
<tr>
<td>Name</td>
<td>null terminated string</td>
</tr>
</tbody>
</table>

Within each file, you can identify a Resource by its unique combination of Type and Number, or by its Name.

**Note: The Resource Name is not guaranteed to be unique.**

**Resource Type**

Resource Type gathers Resources into related groups. Resource Types in the range of 1 to 127 are reserved for use by FairCom. These are special Resources used by the c-treeACE, r-tree Report Generator, the d-tree application builder, c-treeDB, and c-treeACE SQL. Resource Types in the range of 128 to 65536 are also reserved and are explicitly assigned to third party
developers by FairCom. Resource Types numbered 65537 and above are available to any developer and can be assigned as desired.

**Resource Number**

There are no restrictions on assigning Resource Numbers. They separate various Resources within a given Resource Type. When adding a Resource to a file, c-treeACE can assign the next available value for this Resource Type in the referenced table.

**Resource Name**

You can use the Resource Name as a way to access Resources in a file instead of using the Type and Number. This can make application code more readable if you use a symbolic name rather than a pair of numbers. If you are adding a number of Resources to a data file over a period of time you may not know what Resource Number has been used, particularly as c-treeACE can automatically assign the next available Resource Number. Access via Resource Name is simpler than keeping track of what Resource Number has been assigned.

However, use caution when assigning Resource Names. c-treeACE cannot guarantee that each Resource Name is unique within a table. It is possible to add a duplicated Resource name to a table.

Resource Names are optional. They are not required for a Resource. The Resource Name is a null terminated character string of any length. FairCom recommends you do not make them too long. Resource Names starting with the character sequence "FC!" are reserved for use by FairCom. Resource Names starting with with the character sequence "RD!" are reserved for Resources assigned to third party developers by FairCom.

**c-treeDB C++ API - Working with Resources**

Resource operations are usually performed in the following sequence:

1. Instantiate a `CTResource` object by calling one of the `CTResource` constructors.
2. Perform the desired operation such as adding, updating, deleting or reading resources
3. Once the `CTResource` object is no longer needed, destroy the object to release any system resources.

**Constructing Resource Objects**

Before any operations can be performed on resources, you are required to instantiate a resource object. There are three overloaded `CTResource` constructors:

```
CTResource::CTResource(const CTTable& hTable);
```

Construct a `CTResource` object with the resource type and number set to zero and the resource name set to NULL.

```
CTResource(const CTTable& hTable, ULONG type, ULONG number);
```

Construct a `CTResource` object passing the resource type and number. The resource name is set to NULL.

```
CTResource(const CTTable& hTable, ULONG type, ULONG number, const CTString& name);
```
Construct a `CTResource` object passing the resource type and number, as well as the resource name.

**Example**

```c
void DisplayAllResources(const CTTable& hTable)
{
    CTResource* hRes = new CTResource(hTable);
    try
    {
        // get the first resource
        if (hRes->First())
        {
            do
            {
                // display resource type, number and name
                printf("Resource type: %u, number: %u",
                    hRes->GetType(), hRes->GetNumber());
            }
            while (hRes->Next());
        }
    }
    catch (CTException &err)
    {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
    }
    delete hRes;
}
```

**Adding New Resources**

When adding a Resource to a table, a special variable-length record is written to the table, containing the information from the Resource Data Block. This is done even if a data file uses fixed-length records. Every Resource is identified by a unique combination of a Resource Type and a Resource Number. The Resource Number can optionally be assigned by c-tree Plus during the call to `CTResource::Add()` method. In addition, each Resource can be identified by a Resource Name. The Resource Name is not guaranteed to be unique.

The following steps are required to add a new resource to a table:

1. Instantiate a `CTResource` object by calling one of the `CTResource` constructors. You should pass at least the resource type and number. The resource name is optional. If you use a `CTResource` constructor that does not take the resource type and number, you need to call `CTResource::SetType()` and `CTResource::SetNumber()` to set the resource type and number before you add a new resource.

2. Call `CTResource::Add()` method to add the new resource passing the resource data and the length of the data.

3. Once the `CTResource` object is no longer needed, destroy the object to release any system resources.

To add a new resource you must call the following `CTResource` class method:

```c
void CTResource::Add(pVOID data, VRLEN size);
```

`data` is any collection of data that you wish to store as a Resource. This can be a character string, a structure, or any variable type. `size` indicates the number of bytes occupied by `data` parameter value.
The Resource Type must be a value greater than 65536. 0 through 65536 are reserved for FairCom use. If the Resource Number is a value of `CTDB.Assign_Resource_Number` (0xffffffff), c-tree Plus assigns this Resource the next available Resource Number for this Resource Type in the specified data file. The assigned number can be retrieved by calling `CTResource::GetNumber()` method before the resource object is destroyed. The Resource Name is optional. c-tree Plus does not guarantee unique Resource Names. Names starting with the character sequence "FC!" or "RD!", are reserved for FairCom use.

Example

```c
void AddMyResource(const CTable& hTable, ULONG type, const CString& name, PVOID data, VRLEN size)
{
    CResource* hRes = new CResource(hTable, type, CTDB.Assign_Resource_Number, name);
    try
    {
        hRes->Add(data, size);
        printf("Resource added with number %u\n", hRes->GetNumber());
    }
    catch (CTException &err)
    {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
    }
    delete hRes;
}
```

Deleting Resources

The following steps are required to delete a resource from a table:

1. Instantiate a `CResource` object by calling one of the `CResource` constructors. You should pass at least the resource type and number. The resource name is optional. If you use a `CResource` constructor that does not take the resource type and number, you need to call `CResource::SetType()` and `CResource::SetNumber()` to set the resource type and number before you add a new resource.

2. Call `CResource::Delete()` method to delete the resource.

3. Once the `CResource` object is no longer needed, destroy the object to release any system resources.

A resource can be deleted from a table by calling the following `CResource` class method:

```c
void CResource::Delete();
```

Example

```c
void DelMyResource(const CTable& hTable, ULONG type, ULONG number)
{
    CResource* hRes = new CResource(hTable, type, number);
    try
    {
        hRes->Delete();
    }
    catch (CTException &err)
    {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
    }
    delete hRes;
}
Updating Existing Resources

The following steps are required to update a resource:

1. Instantiate a CTResource object by calling one of the CTResource constructors. You should pass at least the resource type and number. The resource name is optional. If you use a CTResource constructor that does not take the resource type and number, you need to call CTResource::SetType() and CTResource::SetNumber() to set the resource type and number before you add a new resource.
2. Call CTResource::Update() method passing the resource data and the size in bytes of the resource data.
3. Once the CTResource object is no longer needed, destroy the object to release any system resources.

An existing resource can be updated by calling the following CTResource method:

```cpp
void Update(pVOID data, VRLEN size);
```

The Resource data is any collection of data that you wish to store as a Resource. It can be a character string, a structure, or any variable type. size indicate the number of bytes occupied by data.

Example

```cpp
void UpdateMyResource(const CTTable& hTable, ULONG type, ULONG number, pVOID data, VRLEN size)
{
    CTResource* hRes = new CTResource(hTable, type, number);
    try
    {
        hRes->Update(data, size);
    }
    catch (CTException &err)
    {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
    }
    delete hRes;
}
```

Reading Resources

Essentially, there are four different methods to read resources from a c-tree Plus table. Read all table resources, read all resources starting past a resource type and number, read a specific resource by specifying the Resource Type and Resource Number, or read a resource by specifying a Resource Name. Described below are examples of each method.

Read all c-tree Plus Resources from a Table

Read all resource starting with the first resource by calling method CTResource::First() and then read the remaining resources by calling method CTResource::Next(). To read the first resource in a table, call the following CTResource method:

```cpp
CTBOOL CTResource::First(CTBOOL lock = NO) const;
```

CTResource::First() retrieve the first resource stored in a table.

- **lock** is used to indicate if the resource should be locked, if it is found.
**CTResource::First()** returns YES if the resource was found or NO if no resource was found. If an error is detected, **CTResource::First()** throws a **CTException** exception.

To read all other resources of a table call the following CTResource method:

```cpp
CTBOOL CTResource::Next(CTBOOL lock = NO) const;
```

**CTResource::Next()** retrieves the next resource stored in a table, after a successful call to **CTResource::First()** is made.

- **lock** is used to indicate if the resource should be locked, if it is found.

**CTResource::Next()** returns YES if the resource was found or NO if no resource was found. If an error is detected, **CTResource::First()** throws a **CTException** exception.

**Example**

```cpp
void DisplayAllResources(const CTTable& hTable)
{
    CTResource* hRes = new CTResource(hTable);

    try
    {
        // get the first resource
        if (hRes->First())
        {
            do
            {
                // display resource type, number and name
                printf("Resource type: %u, number: %u",
                        hRes->GetType(), hRes->GetNumber());
            } while (hRes->Next());
        }
    }
    catch (CTException &err)
    {
        printf("Error %d - %s
        delete hRes;
    }
}
```

**Read all c-tree Plus Resources Given a Starting Point from a Table**

Read all resources starting from a given resource type and number by calling method **CTResource::Next()**. Start by instantiating a resource object setting the resource type and a resource number and then call **CTResource::Next()**. In this case a resource is found if its type is greater or equal to the specified type and its number is greater than the specified number.

**Example**

```cpp
// read resources with type >= type and number > 0
void DisplayResources(const CTTable& hTable, ULONG type)
{
    CTResource* hRes = new CTResource(hTable, type, 0);

    try
    {
        while (hRes->Next())
        {
            printf("Resource type: %u, number: %u",
```
hRes->GetType(), hRes->GetNumber();
}
}

Read a Specific Plus Resources from a Table

Read a specific resource by providing exact resource type and number by calling
CTResource::Find():

CTBOOL Find(ULONG type, ULONG number, CTBOOL lock = NO) const;

Locates and retrieves a resource in a table based on type and number.
- **type** is a value of the resource type.
- **number** is the value of the resource number to be located.
- **lock** is used to indicate if the resource should be locked, if it is found.

CTResource::Find() returns YES if the resource was found or NO if this particular resource was
not found. If an error occurs, CTResource::Find() throws a CTException exception.

Example

```c++
// display a particular resource
void DisplayResource(const CTTable& hTable, ULONG type, ULONG number)
{
    CTResource* hRes = new CTResource(hTable);
    try
    {
        if (hRes->Find(type, number))
        {
            printf("Resource type: %u, number: %u, name: %s\n",
                    hRes->GetType(), hRes->GetNumber(),
                    hRes->GetName().c_str());
        }
        else
            printf("Resource %d,%d not found\n", type, number);
    }
    catch (CTException &err)
    {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
    }
    delete hRes;
}
```

Read a c-tree Plus Resources by Name from a Table

Read a resource by providing its name by calling CTResource::Find():

CTBOOL CTResource::Find(const CTString& name, CTBOOL lock = NO) const;

CTResource::Find() locates and retrieves a resource by name. c-tree Plus cannot guarantee
unique resource names.
- **name** is the resource name being located.
• *lock* should indicate if the resource should locked, if it is found.

CTResource::Find() returns YES if the resource is located or NO if no resource is located. If errors are detected a CTException exception object is thrown.

**Example**

```cpp
// display a particular resource by name
void DisplayResource(const CTTable& hTable, const CTString& name)
{
    CTResource* hRes = new CTResource(hTable);

    try
    {
        if (hRes->Find(type, name))
        {
            printf("Resource type: %u, number: %u, name: %s\n", hRes->GetType(), hRes->GetNumber(), hRes->GetName().c_str());
        }
        else
            printf("Resource %s not found\n", name.c_str());
    }
    catch (CTException &err)
    {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
    }
    delete hRes;
}
```

**Get and Set Resource Properties with the c-treeDB C++ API**

A number of methods are provided to enable the getting and setting of resource object properties. The table bellow describe each function:

<table>
<thead>
<tr>
<th>c-treeDB Resource C++ API Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTResource::GetType</td>
<td>get the resource type</td>
</tr>
<tr>
<td>CTResource::SetType</td>
<td>set the resource type</td>
</tr>
<tr>
<td>CTResource::GetNumber</td>
<td>get the resource number</td>
</tr>
<tr>
<td>CTResource::SetNumber</td>
<td>set the resource number</td>
</tr>
<tr>
<td>CTResource::GetName</td>
<td>get the resource name</td>
</tr>
<tr>
<td>CTResourceSetName</td>
<td>set the resource name</td>
</tr>
<tr>
<td>CTResource::GetDataLength</td>
<td>get the resource data buffer length in bytes</td>
</tr>
<tr>
<td>CTResource::GetData</td>
<td>get a pointer to resource data buffer</td>
</tr>
<tr>
<td>CTResource::GetData</td>
<td>set the resource data buffer</td>
</tr>
</tbody>
</table>

**Resource Locks**

It is expected that you process resource updates without permitting user interactions to occur during the actual update. It is important for performance considerations and c-treeDB assumes that resource entries will be locked only for very short intervals. Do not lock a resource and then...
request user input. Be careful not to include resource updates, (additions, deletions or updates), in long held transactions when the table is under transaction processing control. Locks cannot be released until a transaction either commits or aborts.

c-treeDB resource handling automatically unlocks resources when necessary, however, a resource can be manually unlocked by the user. Use the following function to check if a resource is locked or not:

\[ \text{CTBOOL} \ \text{CTResource::IsLocked() const;} \]

\textbf{CTResource::IsLocked()} returns YES if the resource referenced by the resource handle is locked, otherwise it returns NO. The following method should be used to unlock resources whose locks are no longer necessary:

\[ \text{void} \ \text{CTResource::Unlock();} \]

\textbf{CTResource::Unlock()} releases a lock placed on a resource by one of the following read resource functions: \textbf{CTResource::First()}, \textbf{CTResource::Next()} and \textbf{CTResource::Find()}.

c-treeDB C++ API Resource Locking Example

\begin{verbatim}
if (hRes->IsLocked())
    hRes->Unlock();
\end{verbatim}
3.9 Working with Callbacks

c-treeDB represents a high-level, easy to use API on top of the popular c-treeACE ISAM and c-treeACE Low-Level APIs. c-treeDB is intended as the new standard for c-treeACE programming. When compared to ISAM and low-level APIs, c-treeDB introduced a more formal definition for the structure of data and index files, new concepts such as ROWID and NULL field support, and more specifically a formal definition for each field type supported.

Existing applications’ data and index files, i.e. data created prior to the publication of the c-treeDB API, may have implemented certain field types in ways that may be incompatible with c-treeDB. Field types such as CT_DATE, CT_TIME and CT_TIMES are probably the most common examples of existing data that may be incompatible with c-treeDB.

The c-treeDB Callback feature was implemented to provide developers a means to intercept certain c-treeDB operations and add custom code to manipulate record buffer layouts or change field data on the fly such that the record and field data are compatible with c-treeDB.

To enable a given callback, perform the following:

1. Write your callback functions. Every callback function must have the callback function type. There is no need to implement every single callback type. Implement only the callback functions that are necessary for your logic.
2. Set your callbacks with the ctdbSetCallback() function. For example if you have implemented CTDB_ON_LOGON and CTDB_ON_LOGOUT callback functions, you call the ctdbSetCallback() function once for each callback to the session handle with the appropriate callback type.

You remove a callback from a c-treeDB handle by calling the ctdbClearCallback() function. After removed, a callback for a particular handle will not be called again. You can remove all callbacks from a particular session, database, table or record handle by calling the ctdbClearAllCallback() function.

Callback Function Type

Every c-treeDB callback function must be a function that returns CTDBRET and takes a CTHANDLE parameter. The following typedef declares the function callback type:

typedef CTDBRET (ctdbDECL* ctdbCallbackFunc)(CTHANDLE Handle);

When implementing callbacks you need to create your callback functions with the same type of ctdbCallbackFunc. Below is an example of a c-treeDB session logon callback.

c-treeDB C API Example

CTDBRET ctdbDECL OnSessionLogon(CTHANDLE Handle)
{
    pCTDBSESSION pSession = (pCTDBSESSION)Handle;

    printf("Server: %s\n", pSession->server_name);
    printf("User: %s\n", pSession->user_name);
    printf("Password: %s\n", pSession->user_password);
}
Please refer to the Callback Types (page 186) section below for more information on each available callback type.

### Callback Return Codes

Every callback function must return a `CTDBRET` value. If no errors are detected the callback function must return `CTDBRET_OK`. If a callback is not yet implemented it should return `CTDBRET_NOTIMPLEMENTED`. Please refer to the `ctdbsdk.h` header file for valid values for `CTDBRET`.

### Callback Handle Parameters

The type of handle that is passed to a callback function depends on the type of the callback.

#### Sessions

For these types the handle parameter is a session handle. You can safely typecast the handle parameter to a `pCTDBSESSION` structure pointer. Please refer to the `ctdbsdk.h` header file for the declaration of `CTDBSESSION` structure.

- `CTDB_ON_SESSION_LOGON`
- `CTDB_ON_SESSION_LOGOUT`

#### Databases

For these types the handle parameter is a database handle. You can safely typecast the handle parameter to a `pCTDBDATABASE` structure pointer. Please refer to the `ctdbsdk.h` header file for the declaration of `CTDBDATABASE` structure.

- `CTDB_DATABASE_CONNECT`
- `CTDB_DATABASE_DISCONNECT`

#### Tables

For these types the handle parameter is a table handle. You can safely typecast the handle parameter to a `pCTDBTABLE` structure pointer. Please refer to the `ctdbsdk.h` header file for the declaration of `CTDBTABLE` structure.

- `CTDB_ON_TABLE_OPEN`
- `CTDB_ON_TABLE_CLOSE`
- `CTDB_ON_TABLE_GET_DODA`
- `CTDB_ON_TABLE_GET_SCHEMA`
- `CTDB_ON_TABLE_GET_EXT_INFO`
- `CTDB_ON_TABLE_GET_VTABLE_INFO`
- `CTDB_ON_TABLE_ALTER`
- `CTDB_ON_TABLE_REBUILD`
Records

For these types the handle parameter is a record handle. You can safely typecast the handle parameter to a `pCTDBRECORD` structure pointer. Please refer to the `ctdbsdk.h` header file for the declaration of `CTDBRECORD` structure.

- `CTDB_ON_RECORD_INIT`
- `CTDB_ON_RECORD_RESET`
- `CTDB_ON_RECORD_BEFORE_READ`
- `CTDB_ON_RECORD_AFTER_READ`
- `CTDB_ON_RECORD_BEFORE_BUILD_KEY`
- `CTDB_ON_RECORD_AFTER_BUILD_KEY`
- `CTDB_ON_RECORD_BEFORE_WRITE`
- `CTDB_ON_RECORD_AFTER_WRITE`

Callback Types

When a callback is registered with a c-treeDB handle, you need to specify which callback is being intercepted. The sections below describe each of the callback types that can be intercepted.

**CTDB_ON_SESSION_LOGON**

A `CTDB_ON_SESSION_LOGON` callback is invoked after a successful session logon but before the `ctdbLogon()` function returns. The handle passed as a parameter with this callback is a session handle. You can safely typecast the handle parameter to a `pCTDBSESSION` structure pointer.

The session logon callback can be used as an indication that the session is now active and that database and table processing may occur. This callback is a good place to set session-wide parameters that may be active until a session logout is performed.

**c-treeDB C API Example**

```c
CTDBRET ctdbDECL OnSessionLogon(CTHANDLE Handle)
{
    CTDBRET Retval = CTDBRET_OK;
    pCTDBSESSION pSession = (pCTDBSESSION)Handle;

    if (!pSession)
        Retval = CTDBRET_NOTSESSION;
    else
    {
        /* allocate 100 bytes of memory and keep it in the session local tag */
        pSession->localTag = pSession->OnAlloc(100);
        if (pSession->localTag)
            Retval = CTDBRET_NOMEMORY;
    }
    return Retval;
}
```
**CTDB_ON_SESSION_LOGOUT**

A **CTDB_ON_SESSION_LOGOUT** callback is invoked before a session logout is performed. The handle passed as a parameter with this callback is a session handle and you can safely typecast the handle parameter as a `pCTDBSESSION` structure pointer.

The session logout callback can be used to undo and cleanup operations performed by a session logon callback. No other callbacks calls will be issued after a **CTDB_ON_SESSION_LOGOUT** callback.

c-treeDB C API Example

```c
CTDBRET ctdbDECL OnSessionLogout(CTHANDLE Handle)
{
    CTDBRET Retval = CTDBRET_OK;
    pCTDBSESSION pSession = (pCTDBSESSION)Handle;

    if (!pSession)
        Retval = CTDBRET_NOTSESSION;
    else
    {
        /* release any memory allocated by session logon callback */
        if (pSession->localTag)
        {
            pSession->OnFree(pSession->localTag)
            pSession->localTag = NULL;
        }
    }
    return Retval;
}
```

**CTDB_ON_DATABASE_CONNECT**

A **CTDB_ON_DATABASE_CONNECT** callback is invoked after a successful database connect but before the `ctdbConnect()` function returns. The handle passed as a parameter with this callback is a database handle and you can safely typecast the handle parameter to a `pCTDBDATABASE` structure pointer.

The database connect callback can be used as an indication that the database connection is now active and that table processing may occur. This callback is a good place to set database-wide parameters that may be active until a database disconnect is performed.

Below is an example demonstrating the database connect callback to set table open and table close callbacks.

c-treeDB C API Example

```c
CTDBRET ctdbDECL OnDatabaseConnect(CTHANDLE Handle)
{
    CTDBRET Retval = CTDBRET_OK;
    pCTDBDATABASE pDatabase = (pCTDBDATABASE)Handle;

    if (!pDatabase)
        Retval = CTDBRET_NOTDATABASE;
    else
    {
        Retval = ctdbSetCallback(Handle, CTDB_ON_TABLE_OPEN, onTableOpen);
        if (Retval == CTDBRET_OK)
```
Retval = ctdbSetCallback(Handle, CTDB_ON_TABLE_CLOSE, onTableClose);
}
return Retval;
}

CTDB_ON_DATABASE_DISCONNECT

A CTDB_ON_DATABASE_DISCONNECT callback is invoked just before a database disconnect is performed. The handle passed as a parameter with this callback is a database handle and you can safely typecast the handle parameter to a pCTDBDATABASE structure pointer.

The database disconnect callback may be used to undo and cleanup any operations performed by a database connect callback.

Below is an example demonstrating how to clear the callbacks established by the database connect callback.

c-treeDB C API Example

CTDBRET ctdbDECL OnDatabaseDisconnect(CTHANDLE Handle)
{
    CTDBRET Retval = CTDBRET_OK;
pCTDBDATABASE pDatabase = (pCTDBDATABASE)Handle;

    if (!pDatabase)
        Retval = CTDBRET_NOTDATABASE;
    else
    {
        Retval = ctdbClearCallback(Handle, CTDB_ON_TABLE_OPEN);
        if (Retval == CTDBRET_OK)
           Retval = ctdbClearCallback(Handle, CTDB_ON_TABLE_CLOSE);
    }
    return Retval;
}

CTDB_ON_TABLE_OPEN

A CTDB_ON_TABLE_OPEN callback is invoked after the c-treeACE data and index files are open, but before ctdbOpenTable() returns. The handle passed as a parameter with this callback is a table handle and you can safely typecast the handle parameter to a pCTDBTABLE structure pointer.

The open table callback can be used as an indication that the table is active and that table and records operations may occur.

Below is an example demonstrating how to obtain the record callbacks when the table open callback is called.

c-treeDB C API Example

typedef struct
{
    pCTDBSESSION  pSession;
pCTDBTABLE    pTable;
pDATOBJ       dodaptr;
VRLEN         doda_size;
VRLEN         dodacount;
NINT          fixlen;
} LOCALTABLE, ctMEM* pLOCALTABLE;
CTDBRET ctdbDECL OnTableOpen(CHANDLE Handle)
{
    CTDBRET Retval = CTDBRET_OK;
pCTDBTABLE pTable = (pCTDBTABLE)Handle;
pCTDBSESSION pSession;
pLOCALTABLE pLocal = NULL;

    /* check the table handle */
    if (!pTable)
    {
        Retval = CTDBRET_NOTTABLE;
goto Exit;
    }
pSession = (pCTDBSESSION)pTable->pSession;

    /* check the table name for our table */
    if (strcmp(pTable->name, "table") != 0)
    {
        /* this is not the table we are looking for */
goto Exit;
    }

    /* allocate the local data for the table handle */
pLocal = (pLOCALTABLE)pSession->onAlloc(sizeof(LOCALTABLE));
    if (!pLocal)
    {
        Retval = CTDBRET_NOMEMORY;
goto Exit;
    }

    /* initialize the local table data */
pLocal->pSession = pSession;
pLocal->pTable = pTable;
pLocal->dodaptr = NULL;
pLocal->doda_size = 0;
pLocal->doda_count = 0;
pTable->localTag = (pVOID)pLocal;
pLocal = NULL;

    /* set the other callbacks for this table */
    if ((Retval = ctdbSetCallback(Handle, CTDB_ON_TABLE_OPEN, OnTableOpen)) != CTDBRET_OK) goto Exit;
    if ((Retval = ctdbSetCallback(Handle, CTDB_ON_TABLE_GET_DODA, OnTableGetDoda)) != CTDBRET_OK) goto Exit;
    if ((Retval = ctdbSetCallback(Handle, CTDB_ON_TABLE_GET_RECLEN, OnTableGetReclen)) != CTDBRET_OK) goto Exit;
    if ((Retval = ctdbSetCallback(Handle, CTDB_ON_TABLE_CLOSE, OnTableClose)) != CTDBRET_OK) goto Exit;
    if ((Retval = ctdbSetCallback(Handle, CTDB_ON_RECORD_INIT, OnRecordInit)) != CTDBRET_OK) goto Exit;
    if ((Retval = ctdbSetCallback(Handle, CTDB_ON_RECORD_RESET, OnRecordReset)) != CTDBRET_OK) goto Exit;
    if ((Retval = ctdbSetCallback(Handle, CTDB_ON_RECORD_BEFORE_READ, OnRecordBeforeRead)) != CTDBRET_OK) goto Exit;
    if ((Retval = ctdbSetCallback(Handle, CTDB_ON_RECORD_AFTER_READ, OnRecordAfterRead)) != CTDBRET_OK) goto Exit;
    if ((Retval = ctdbSetCallback(Handle, CTDB_ON_RECORD_BEFORE_BUILD_KEY, OnRecordBeforeBuildKey)) != CTDBRET_OK) goto Exit;
    if ((Retval = ctdbSetCallback(Handle, CTDB_ON_RECORD_AFTER_BUILD_KEY, OnRecordAfterBuildKey)) != CTDBRET_OK) goto Exit;
}

Exit:
return Retval;
}
if ((Retval = ctdbSetCallback(Handle, CTDB_ON_RECORD_BEFORE_WRITE, OnRecordBeforeWrite)) != CTDBRET_OK) goto Exit;
if ((Retval = ctdbSetCallback(Handle, CTDB_ON_RECORD_AFTER_WRITE, OnRecordAfterWrite)) != CTDBRET_OK) goto Exit;

/* set the callback for any records already allocated */
if (pTable->records) {
    NINT i;
    for (i = 0; i < pTable->records->count; i++) {
        CTHANDLE recHandle = (CTHANDLE)_ctdbListItem(pTable->records, i);
        if (recHandle) {
            if ((Retval = ctdbSetCallback(recHandle, CTDB_ON_RECORD_INIT, OnRecordInit)) != CTDBRET_OK) goto Exit;
            if ((Retval = ctdbSetCallback(recHandle, CTDB_ON_RECORD_RESET, OnRecordReset)) != CTDBRET_OK) goto Exit;
            if ((Retval = ctdbSetCallback(recHandle, CTDB_ON_RECORD_BEFORE_READ, OnRecordBeforeRead)) != CTDBRET_OK) goto Exit;
            if ((Retval = ctdbSetCallback(recHandle, CTDB_ON_RECORD_AFTER_READ, OnRecordAfterRead)) != CTDBRET_OK) goto Exit;
            if ((Retval = ctdbSetCallback(recHandle, CTDB_ON_RECORD_BEFORE_BUILD_KEY, OnRecordBeforeBuildKey)) != CTDBRET_OK) goto Exit;
            if ((Retval = ctdbSetCallback(recHandle, CTDB_ON_RECORD_AFTER_BUILD_KEY, OnRecordAfterBuildKey)) != CTDBRET_OK) goto Exit;
            if ((Retval = ctdbSetCallback(recHandle, CTDB_ON_RECORD_BEFORE_WRITE, OnRecordBeforeWrite)) != CTDBRET_OK) goto Exit;
            if ((Retval = ctdbSetCallback(recHandle, CTDB_ON_RECORD_AFTER_WRITE, OnRecordAfterWrite)) != CTDBRET_OK) goto Exit;
        }
    }
}

Exit:
if (pLocal) {
    if (pSession) pSession->onFree(pLocal);
    return Retval;
}

CTDB_ON_TABLE_CLOSE

A CTDB_ON_TABLE_CLOSE callback is invoked before the c-treeACE data and index files are closed, but before the ctdbCloseTable() function returns. The handle passed as a parameter with this callback is a table handle and you can safely typecast the handle parameter to a pCTDBTABLE structure pointer. The close table callback may be used to undo and cleanup operations performed by the table open callback.

c-treeDB C API Example

CTDBRET ctdbDECL OnTableClose(CTHANDLE Handle) {
pCTDBTABLE pTable = (pCTDBTABLE)Handle;
if (pTable && pTable->localTag) {

pCTDBSESSION pSession = (pCTDBSESSION)pTable->pSession;
pLOCALTABLE pLocal = (pLOCALTABLE)pTable->localTag;

if (pLocal)
{
    if (pLocal->dodaptr)
    {
        pSession->onFree(pLocal->dodaptr);
        pSession->onFree(pTable->localTag);
    }
    pTable->localTag = NULL;
}
return CTDBRET_OK;

CTDB_ON_TABLE_GET_SCHEMA

A CTDB_ON_TABLE_GET_SCHEMA callback is invoked after the c-treeACE data and index files are open and the CTDB_ON_TABLE_OPEN callback is invoked. This callback is also invoked after the c-treeACE data file record schema object is loaded into the table handle.

You may use the CTDB_ON_TABLE_GET_SCHEMA callback event to modify c-treeACE schema information stored in the table handle. You should typecast the table handle to a pCTDBTABLE structure pointer. The following CTDBTABLE structure members keep schema information:

<table>
<thead>
<tr>
<th>CTDBTABLE Structure Member</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>pConvMap schemaptr</td>
<td>c-tree ConvMap structure pointer</td>
</tr>
<tr>
<td>VRLEN schema_size</td>
<td>size in bytes of allocated ConvMap structure</td>
</tr>
</tbody>
</table>

At the point a CTDB_ON_TABLE_GET_SCHEMA callback is invoked, the table’s schema has already been loaded and stored in the schemaptr member of CTDBTABLE structure. The schema_size member contains the allocated size of schemaptr.

c-treeDB C API Example

CTDBRET ctdbDECL OnTableGetSchema(CTHANDLE Handle)
{
    CTDBRET Retval = CTDBRET_OK;
pCTDBTABLE pTable = (pCTDBTABLE)Handle;

    if (!pTable)
        Retval = CTDBRET_NOTTABLE;
    else
    {
        /* change the field padding byte */
        if (pTable->schemaptr)
            pTable->schemaptr->padding = ' ';  
        return Retval;
    }
}

CTDB_ON_TABLE_GET_DODA

A CTDB_ON_TABLE_GET_DODA callback is invoked after c-treeACE data and index files are open and the CTDB_ON_TABLE_OPEN and CTDB_ON_TABLE_GET_SCHEMA callbacks have
been invoked. This callback is also invoked after the c-treeACE data file record schema and DODA object are loaded into the table handle.

You can use the `CTDB_ON_TABLE_GET_DODA` callback event to modify the c-treeACE DODA stored in the table handle. You should typecast the table handle to a `pCTDBTABLE` structure pointer. The following `CTDBTABLE` structure members keep DODA information:

<table>
<thead>
<tr>
<th>CTDBTABLE Structure Member</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDATOBJ dodaptr</td>
<td>pointer to c-tree DODA</td>
</tr>
<tr>
<td>VRLEN dodacount</td>
<td>number of fields in DODA</td>
</tr>
<tr>
<td>VRLEN doda_size</td>
<td>size in bytes of allocated DODA</td>
</tr>
</tbody>
</table>

By the time a `CTDB_ON_TABLE_GET_DODA` callback is invoked, the table’s DODA has already been loaded and stored in the `dodaptr` member of `CTDBTABLE` structure. The `doda_size` member contains the allocated size of `dodaptr` and `dodacount` keep count of the number of fields described by `dodaptr`.

The example below shows how to use the `CTDB_ON_TABLE_GET_DODA` callback to modify the DODA information.

**c-treeDB C API Example**

```c
static DATOBJ newdoda[4] =
{
    {"id", (pTEXT)0, CT_INT4U, 4},
    {"who", (pTEXT)4, CT_INT2, 2},
    {"when", (pTEXT)8, CT_TIMES, 8},
    {"text", (pTEXT)16, CT_STRING, 0}
};

CTDBRET ctdbDECL OnTableGetDoda(CTHANDLE Handle)
{
    CTDBRET Retval = CTDBRET_OK;
    pCTDBTABLE pTable = (pCTDBTABLE)Handle;
    NINT i;

    /* check the table handle */
    if (!pTable)
    {
        Retval = CTDBRET_NOTTABLE;
        goto Exit;
    }

    /* fix the DODA */
    for (i = 0; i < pLocal->dodacount; i++)
    {
        pTable->dodaptr[i].fadr = newdoda[i].fadr;
        pTable->dodaptr[i].flen = newdoda[i].flen;
        pTable->dodaptr[i].ftype = newdoda[i].ftype;
    }

    /* fix the IFIL index segment */
    if (pTable->ifilptr && pTable->ifilptr->ix && pTable->ifilptr->ix[0].seg)
    {
        pTable->ifilptr->ix[0].seg->soffset = 0;
        pTable->ifilptr->ix[0].seg->slength = 4;
    }

exit:
}
```
CTDB_ON_TABLE_GET_RECLEN

A CTDB_ON_TABLE_GET_RECLEN callback is invoked after c-treeACE data and index files are open and the T and CTDB_ON_TABLE_GET_SCHEMA and CTDB_ON_TABLE_GET_DODA callbacks have been invoked. This callback is also invoked after the c-treeACE data file record schema and DODA object are loaded into the table handle.

You can use the CTDB_ON_TABLE_GET_RECLEN callback event to control the length, in bytes, of the fixed portion of the record buffer. The handle passed to this callback is always a table handle and you can safely typecast the table handle to a pCTDBTABLE structure pointer.

The following CTDBTABLE structure members keep the length of the fixed portion of the record buffer:

<table>
<thead>
<tr>
<th>CTDBTABLE Structure Member</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VRLEN fixreclen</td>
<td>length of fixed portion of record buffer</td>
</tr>
</tbody>
</table>

By the time CTDB_ON_TABLE_GET_RECLEN callback is invoked, the fixreclen member of CTDBTABLE structure has already been calculated. You can use this event to modify the length of the fixed portion of the record buffer.

c-treeDB C API Example

CTDBRET ctdbDECL OnTableGetRecLen(CTHANDLE Handle)
{
    pCTDBTABLE pTable = (pCTDBTABLE)Handle;

    if(pTable)
    {
        pTable->fixreclen = 16;
        return CTDBRET_OK;
    }
}

CTDB_ON_TABLE_GET_EXT_INFO

C-treeDB maintains extended field information that is not present in the DODA object, but is necessary for optimal c-treeACE SQL operation. Each DODA entry describes the field name, the field offset in the record buffer, the field type and length, but for optimal SQL operation we also need additional information such as the field precision, scale and if the field allows NULL values.

A field precision is the total number of digits necessary to represent the value, while the field scale is the number of digits to the left of the decimal point. For example CT_NUMBER fields have a precision of 32 and scale from 0 to 32. CT_INT4 fields have precision of 10 and scale of 0. A NULL value for a field indicates that the field has no value (i.e., no value was assigned to the field when the record was written to the table.)

A CTDB_ON_TABLE_GET_EXT_INFO callback is invoked after a c-treeACE data and index files are open and the CTDB_ON_TABLE_OPEN and CTDB_ON_TABLE_GET_SCHEMA, CTDB_ON_TABLE_GET_DODA and CTDB_ON_TABLE_GET_RECLEN callbacks have been invoked.
invoked. This callback is also invoked after the c-treeDB open table code has tried to load the extended field information resource.

You can use the `CTDB_ON_TABLE_GET_EXT_INFO` callback event to add new, or modify existing, extended field information kept by the table handle. See the example below to see how to modify the extended field information. The handle passed to this callback is always a table handle and you can safely typecast the table handle to a `pCTDBTABLE` structure pointer.

**c-treeDB C API Example**

```c
struct extinfo_tag
{
    CTBOOL  nonull; /* YES = no NULL values accepted */
    COUNT   fprec;  /* field precision */
    COUNT   fscale; /* field scale */
} extinfo[] =
{
    {YES, 10, 0},   /* "id", 0, CT_INT4, 4 */
    {NO,  5, 0},   /* "who", 4, CT_INT2, 2 */
    {NO,  8, 0},   /* "when", 8, CT_TIMES, 8 */
    {NO,  0, 0}    /* "text", 16, CT_STRING, 0 */
};

CTDBRET ctdbDECL OnTableGetExtInfo(CTHANDLE Handle)
{
    CTDBRET Retval = CTDBRET_OK;
    pCTDBTABLE pTable = (pCTDBTABLE)Handle;
    if (!pTable)
        Retval = CTDBRET_NOTTABLE;
    else
    {
        NINT count = (pTable->fields) ? pTable->fields->count : 0;
        NINT i;
        for (i = 0; i < count; i++)
        {
            pCTDBFIELD pField = (pCTDBFIELD)pTable->fields->list[i];
            if (pField)
            {
                pField->nonnull = extinfo[i].nonnull;
                pField->fprec  = extinfo[i].fprec;
                pField->fscale = extinfo[i].fscale;
            }
        }
    }
    return Retval;
}
```

`CTDB_ON_TABLE_GET_VTABLE_INFO` is called when c-treeDB looks for the virtual table resource, typically at the time of a virtual table open or when calling `ctdbGetVTableInfoFromTable()`. This callback should be implemented when using alternative (to the resource as defined) methods of tracking virtual table definitions. (For example, c-treeRTG COBOL Edition support implements this callback.)
**CTDB_ON_TABLE.Alter**

A `CTDB_ON_TABLE.Alter` callback is invoked at the beginning of the `ctdbAlterTable` function call, before any operations are performed on the table. This callback should be used to indicate if alter table operations are to be performed on a particular table.

If the alter table operation is allowed, then the callback function must return a `CTDBRET_OK` value. On the other hand, if alter table operations are not to be allowed, the callback function must return an error code. In case of error, the suggested error code is `CTDBRET_NOT_SUPPORTED`.

The handle passed as a parameter with this callback is a table handle and you can safely typecast the handle parameter to a `pCTDBTABLE` structure pointer.

Below demonstrates an `AlterTable()` not allowed if table name is "mytable".

**c-treeDB C API Example**

```c
CTDBRET ctdbDECL OnTableAlter(CTHANDLE Handle)
{
    CTDBRET Retval = CTDBRET_OK;
    pCTDBTABLE pTable = (pCTDBTABLE)Handle;

    if (pTable && strcmp(pTable->name, "mytable") == 0)
    {
        /* if table is 'mytable' alter table not allowed */
        Retval = CTDBRET_NOT_SUPPORTED;
    }
    return Retval;
}
```

**CTDB_ON_TABLE_REBUILD**

The c-treeDB alter table function was designed and implemented to allow the modification of table properties after it was created and possibly already populated with data.

Depending on which properties are modified, a full table rebuild may be necessary. A full table rebuild is accomplished by creating a temporary table with the correct properties, considering all changes made to the table, and then moving all of the records from the original table to the new temporary table. Once all of the data records have been moved, the original table is deleted and the temporary table renamed with the name of the original table.

During the full table rebuild phase of moving records, a `CTDB_ON_TABLE_REBUILD` callback is invoked for every percentage point of progress. The progress percentage is calculated by dividing the number of records moved by the total number of records in a table. The percentage value is stored in the table handle and the callback function can access this value to display, for example, a progress report during lengthy alter table operations.

If a `CTDB_ON_TABLE_REBUILD` callback returns a value other than `CTDBRET_OK`, the rebuild process will be aborted, and the alter table function will return the value.

The handle passed as a parameter with this callback is a table handle and you can safely typecast the handle parameter to a `pCTDBTABLE` structure pointer. You can access the percentage counter by accessing the following `CTDBTABLE` structure member:
Before any records are moved, the `CTDB_ON_TABLE_REBUILD` callback is called with `rebuild_perc` set to zero to indicate that the operation is about to start. After all records are copied, `CTDB_ON_TABLE_REBUILD` is called again, this time with `rebuild_perc` set to 100 to indicate the end of rebuild.

Since `CTDB_ON_TABLE_REBUILD` callback is called inside the alter table’s record moving loop, care must be taken with the implementation of the callback since it may adversely affect the performance of the alter table operation.

You can also call the `ctdbGetRebuildProgress()` function to retrieve the table rebuild percentage counter.

**c-treeDB C API Example**

```c
CTDBRET ctdbDECL OnTableRebuild(CTHANDLE Handle)
{
    pCTDBTABLE pTable = (pCTDBTABLE)Handle;

    /* display a dot for every 5% rebuild completed */
    if (pTable)
    {
        if (pTable->rebuild_perc > 0 && (pTable->rebuild_perc % 5) == 0)
            printf(".");

        return CTDBRET_OK;
    }
}
```

**CTDB_ON_RECORD_INIT**

A `CTDB_ON_RECORD_INIT` callback is invoked when the record is initialized because it is being used for the first time or after an alter table. The handle passed as a parameter with this callback is a record handle and you can safely typecast the handle parameter to a `pCTDBRECORD` structure pointer.

The record init callback can be used as an indication that the record handle is becoming active and records operations may occur from this point.

The following `CTDBRECORD` structure members keep information regarding the record buffer kept by the record handle:

<table>
<thead>
<tr>
<th>CTDDBRECORD Structure Member</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pUTEXT recbuf</code></td>
<td>record buffer</td>
</tr>
<tr>
<td><code>VRLEN recbuf_size</code></td>
<td>size in bytes of memory allocated for record buffer</td>
</tr>
<tr>
<td><code>VRLEN recbuf_len</code></td>
<td>actual length of data in record buffer</td>
</tr>
</tbody>
</table>

**c-treeDB C API Example**

```c
CTDBRET ctdbDECL OnRecordInit(CTHANDLE Handle)
{
    
```
CTDBRET Retval = CTDBRET_OK;
pCTDBRECORD pRecord = (pCTDBRECORD)Handle;

if (!pRecord)
    Retval = CTDBRET_NOTRECORD;
else
{
    /* allocate a parallel record buffer and store it in the localTag */
    pRecord->localTag = pRecord->pSession ?
        pRecord->pSession->onAlloc(pRecord->recbuf_size) : NULL;
    if (!pRecord->localTag)
        Retval = CTDBRET_OK;
}
return Retval;

CTDB_ON_RECORD_RESET

A CTDB_ON_RECORD_RESET callback is invoked when a record buffer is going inactive because the table is being closed. The handle passed as a parameter with this callback is a record handle and you can safely typecast the handle parameter to a pCTDBRECORD structure pointer.

The following CTDBRECORD structure members keep information regarding the record buffer kept by the record handle:

<table>
<thead>
<tr>
<th>CTDBRECORD Structure Member</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>pUTEXT recbuf</td>
<td>record buffer</td>
</tr>
<tr>
<td>VRLEN recbuf_size</td>
<td>size in bytes of memory allocated for record buffer</td>
</tr>
<tr>
<td>VRLEN recbuf_len</td>
<td>actual length of data in record buffer</td>
</tr>
</tbody>
</table>

c-treeDB C API Example

CTDBRET ctdbDECL OnRecordReset(CTHANDLE Handle)
{
    CTDBRET Retval = CTDBRET_OK;
    pCTDBRECORD pRecord = (pCTDBRECORD)Handle;

    if (!pRecord)
        Retval = CTDBRET_NOTRECORD;
    else
    {
        /* release memory allocated for the localTag */
        if (pRecord->localTag)
            pRecord->pSession->onFree(pRecord->localTag);
        pRecord->localTag = NULL;
    }
    return Retval;
}

CTDB_ON_RECORD_BEFORE_READ

A CTDB_ON_RECORD_BEFORE_READ callback is invoked just before a record is read by one of the c-treeACE record reading functions. This callback is useful to prepare a record buffer just before data is read.
The handle passed as parameter for this callback is a record handle and you can safely typecast the handle parameter to a `pCTDBRECORD` structure pointer.

The following `CTDBRECORD` structure members keep information regarding the record buffer kept by the record handle:

<table>
<thead>
<tr>
<th>CTDBRECORD Structure Member</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pUTEXT recbuf</code></td>
<td>record buffer</td>
</tr>
<tr>
<td><code>VRLEN recbuf_size</code></td>
<td>size in bytes of memory allocated for record buffer</td>
</tr>
<tr>
<td><code>VRLEN recbuf_len</code></td>
<td>actual length of data in record buffer</td>
</tr>
</tbody>
</table>

**c-treeDB C API Example**

```c
CTDBRET ctdbDECL OnRecordBeforeRead(CTHANDLE Handle)
{
    CTDBRET Retval = CTDBRET_OK;
    pCTDBRECORD pRecord = (pCTDBRECORD)Handle;

    if (!pRecord)
        Retval = CTDBRET_NOTRECORD;
    else
    {
        /* swap the record buffer with the localTag */
        pUTEXT ptr = pRecord->recbuf;
        pRecord->recbuf = pRecord->localTag;
        pRecord->localTag = ptr;
    }
    return Retval;
}
```

**CTDB_ON_RECORD_AFTER_READ**

A `CTDB_ON_RECORD_AFTER_READ` callback is invoked just after a record is read by one of the c-treeACE record reading functions. This callback is useful to adjust a record buffer just after data is read.

The handle passed as parameter for this callback is a record handle and you can safely typecast the handle parameter to a `pCTDBRECORD` structure pointer.

The following `CTDBRECORD` structure members keep information regarding the record buffer kept by the record handle:

<table>
<thead>
<tr>
<th>CTDBRECORD Structure Member</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pUTEXT recbuf</code></td>
<td>record buffer</td>
</tr>
<tr>
<td><code>VRLEN recbuf_size</code></td>
<td>size in bytes of memory allocated for record buffer</td>
</tr>
<tr>
<td><code>VRLEN recbuf_len</code></td>
<td>actual length of data in record buffer</td>
</tr>
</tbody>
</table>

**c-treeDB C API Example**

```c
CTDBRET ctdbDECL OnRecordAfterRead(CTHANDLE Handle)
{
    
```
CTDBRET Retval = CTDBRET_OK;
pCTDBRECORD pRecord = (pCTDBRECORD)Handle;

if (!pRecord)
    Retval = CTDBRET_NOTRECORD;
else
{
    /* swap the record buffer with the localTag */
    pUTEXT recptr = pRecord->recbuf;
    int i;

    pRecord->recbuf = pRecord->localTag;
    pRecord->localTag = (pVOID)recptr;

    /* make any CT_DATE field compatible with c-treeDB */
    for (i = 0; (i < pRecord->fields_count && Retval == CTDBRET_OK); i++)
    {
        if (pRecord->fields[i].ftype == CT_DATE)
        {
            time_t* t;
            struct tm* ptr;
            CTDATE date;

            /* get the non standard date into t */
            t = (time_t*)&recptr[pRecord->fields[i].offset];

            /* convert a C time_t date into CT_DATE */
            if ((ptr = gmtime(t)) != NULL)
            {
                if ((Retval = ctdbDatePack(&date, (ptr->tm_year + 1900), (ptr->tm_mon + 1),
                                      ptr->tm_mday)) == CTDBRET_OK)
                {
                    /* put converted CTDB date back into record buffer */
                    memcpy(&pRecord->recptr[pRecord->fields[i].offset], &date,
                           sizeof(CTDATE));
                }
                else
                    Retval = CTDBRET_INVDATE;
            }
        }
    }

    return Retval;
}

CTDB_ON_RECORD_BEFORE_BUILD_KEY

A CTDB_ON_RECORD_BEFORE_BUILD_KEY callback is invoked just before a call to
c-treeACE BuildKey() function. The c-treeACE BuildKey() function perform key segment
translation from record buffer to target key. This callback is useful to prepare a record buffer just
before an index key is built.

The handle passed as parameter for this callback is a record handle and you can safely typecast
the handle parameter to a pCTDBRECORD structure pointer.

The following CTDDBRECORD structure members keep information regarding the record buffer
kept by the record handle:
<table>
<thead>
<tr>
<th>CTDBRECORD Structure Member</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>pUTEXT recbuf</td>
<td>record buffer</td>
</tr>
<tr>
<td>VRLEN recbuf_size</td>
<td>size in bytes of memory allocated for record buffer</td>
</tr>
<tr>
<td>VRLEN recbuf_len</td>
<td>actual length of data in record buffer</td>
</tr>
</tbody>
</table>

**c-treeDB C API Example**

```c
CTDBRET ctdbDECL OnRecordBeforeBuildKey(CTHANDLE Handle)
{
    CTDBRET Retval = CTDBRET_OK;
pCTDBRECORD pRecord = (pCTDBRECORD)Handle;

    if (!pRecord)
        Retval = CTDBRET_NOTRECORD;
    else
    {
        /* change any CT_DATE dates to C time_t dates */
pUTEXT recptr = pRecord->recbuf;
        int i;
        for (i = 0; i < pRecord->fields_count && Retval == CTDBRET_OK; i++)
        {
            if (pRecord->fields[i].ftype == CT_DATE)
            {
                struct tm m;
                CDATE* dateptr;

                /* get the CTDB date */
dateptr = (CTDATE*)&recptr[pRecord->fields[i].offset];

                /* convert a CT_DATE date to a C time_t date */
                if ((Retval = ctdbDateUnpack(*dateptr, &m.tm_year, &m.tm_mon, &m.tm_mday)) == CTDBRET_OK)
                {
                    time_t* tptr = (time_t*)&recptr[pRecord->fields[i].offset];

                    m.tm_year = 1900;
                    m.tm_mon--;
                    m.tm_sec = 0;
                    m.tm_min = 0;
                    m.tm_hour = 0;
                    m.tm_isdst = 0;
                    *tptr = mktime(&m);
                }
            }
        }

        return Retval;
    }
}

**CTDB_ON_RECORD_AFTER_BUILD_KEY**

A **CTDB_ON_RECORD_AFTER_BUILD_KEY** callback is invoked just after a call to the c-treeACE **BuildKey()** function. The c-treeACE **BuildKey()** function performs key segment translation from record buffer to target key. This callback is useful to adjust a record buffer just after a target key was built.
The handle passed as a parameter with this callback is a record handle and you can safely typecast the handle parameter to a `pCTDBRECORD` structure pointer.

The following `CTDBRECORD` structure members keep information regarding the record buffer kept by the record handle:

<table>
<thead>
<tr>
<th>CTDBRECORD Structure Member</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pUTEXT recbuf</code></td>
<td>record buffer</td>
</tr>
<tr>
<td><code>VRLEN recbuf_size</code></td>
<td>size in bytes of memory allocated for record buffer</td>
</tr>
<tr>
<td><code>VRLEN recbuf_len</code></td>
<td>actual length of data in record buffer</td>
</tr>
</tbody>
</table>

**c-treeDB C API Example**

```c
CTDBRET OnRecordAfterBuildKey(CTHANDLE Handle)
{
    CTDBRET Retval = CTDBRET_OK;
    pCTDBRECORD pRecord = (pCTDBRECORD)Handle;

    if (!pRecord)
        Retval = CTDBRET_NOTRECORD;
    else
    {
        /* change any CT_DATE dates to C time_t dates */
        pUTEXT recptr = pRecord->recbuf;
        int i;

        for (i = 0; (i < pRecord->fields_count && Retval == CTDBRET_OK); i++)
        {
            if (pRecord->fields[i].ftype == CT_DATE)
            {
                struct tm m;
                CDATE* dateptr;

                /* get the CTDB date */
                dateptr = (CTDATE*)&recptr[pRecord->fields[i].offset];

                /* convert a CT_DATE date to a C time_t date */
                if ((Retval = ctdbDateUnpack(*dateptr, &m.tm_year, &m.tm_mon, &m.tm_mday)) == CTDBRET_OK)
                {
                    time_t* tptr = (time_t*)&recptr[pRecord->fields[i].offset];

                    m.tm_year = 1900;
                    m.tm_mon = 0;
                    m.tm_sec = 0;
                    m.tm_min = 0;
                    m.tm_hour = 0;
                    m.tm_isdst = 0;
                    *tptr = mktime(&m);
                }
            }
        }
    }

    return Retval;
}
```
CTDB_ON_RECORD_BEFORE_WRITE

A CTDB_ON_RECORD_BEFORE_WRITE callback is invoked just before a record is written with one of the c-treeACE record writing or updating functions. This callback is useful to prepare a record buffer just before data is written to disk.

The handle passed as a parameter with this callback is a record handle and you can safely typecast the handle parameter to a pCTDBRECORD structure pointer.

The following CTDBRECORD structure members keep information regarding the record buffer kept by the record handle:

<table>
<thead>
<tr>
<th>CTDBRECORD Structure Member</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>pUTEXT recbuf</td>
<td>record buffer</td>
</tr>
<tr>
<td>VRLEN recbuf_size</td>
<td>size in bytes of memory allocated for record buffer</td>
</tr>
<tr>
<td>VRLEN recbuf_len</td>
<td>actual length of data in record buffer</td>
</tr>
</tbody>
</table>

c-treeDB C API Example

CTDBRET ctdbDECL OnRecordBeforeWrite(CTHANDLE Handle)
{
    CTDBRET Retval = CTDBRET_OK;
    pCTDBRECORD pRecord = (pCTDBRECORD)Handle;
    if (!pRecord)
        Retval = CTDBRET_NOTRECORD;
    else
    {
        /* change any CT_DATE dates to C time_t dates */
        pUTEXT recptr = pRecord->recbuf;
        int i;
        for (i = 0; (i < pRecord->fields_count && Retval == CTDBRET_OK); i++)
        {
            if (pRecord->fields[i].ftype == CT_DATE)
            {
                struct tm m;
                CTDATE* dateptr;
                /* get the CTDB date */
                dateptr = (CTDATE*)&recptr[pRecord->fields[i].offset];
                /* convert a CT_DATE date to a C time_t date */
                if ((Retval = ctdbDateUnpack(*dateptr, &m.tm_year, &m.tm_mon, &m.tm_mday)) == CTDBRET_OK)
                {
                    time_t* tptr = (time_t*)&recptr[pRecord->fields[i].offset];
                    m.tm_year = 1900;
                    m.tm_mon--;
                    m.tm_sec = 0;
                    m.tm_min = 0;
                    m.tm_hour = 0;
                    m.tm_isdst = 0;
                    *tptr = mktime(&m);
                }
            }
        }
    }
}
CTDB_ON_RECORD_AFTER_WRITE

A CTDB_ON_RECORD_AFTER_WRITE callback is invoked just after a record is written with one of the c-treeACE record writing or updating functions. This callback is useful to adjust a record buffer just after data is written to disk.

The handle passed as a parameter with this callback is a record handle and you can safely typecast the handle parameter to a pCTDBRECORD structure pointer.

The following CTDBRECORD structure members keep information regarding the record buffer kept by the record handle:

<table>
<thead>
<tr>
<th>CTDBRECORD Structure Member</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>pUTEXT recbuf</td>
<td>record buffer</td>
</tr>
<tr>
<td>VRLEN recbuf_size</td>
<td>size in bytes of memory allocated for record buffer</td>
</tr>
<tr>
<td>VRLEN recbuf_len</td>
<td>actual length of data in record buffer</td>
</tr>
</tbody>
</table>

c-treeDB C API Example

CTDBRET ctdbDECL OnRecordAfterWrite(CTHANDLE Handle)
{  
    CTDBRET Retval = CTDBRET_OK;  
    pCTDBRECORD pRecord = (pCTDBRECORD)Handle;  
    
    if (!pRecord)  
        Retval = CTDBRET_NOTRECORD;  
    else  
    {  
        /* change any CT_DATE dates to C time_t dates */  
        pUTEXT recptr = pRecord->recbuf;  
        int i;  
        for (i = 0; (i < pRecord->fields_count && Retval == CTDBRET_OK); i++)  
        {  
            if (pRecord->fields[i].ftype == CT_DATE)  
            {  
                struct tm m;  
                CDATE* dateptr;  
                /* get the CTDB date */  
                dateptr = (CTDATE*)&recptr[pRecord->fields[i].offset];  
                
                /* convert a CT_DATE date to a C time_t date */  
                if ((Retval = ctdbDateUnpack(*dateptr, &m.tm_year, &m.tm_mon, &m.tm_mday)) ==  
                    CTDBRET_OK)  
                {  
                    time_t* tptr = (time_t*)&recptr[pRecord->fields[i].offset];  
                    m.tm_year -= 1900;  
                    m.tm_mon--;  
                }  
            }  
        }  
    }  
}
CTDB_ON_RECORD_MAPTOPARENT

Called only if the child table has the field mapping resource activated. Overrides the field mapping information, that is, if the callback function pointers are set, then the field mapping information stored in the child table resource is ignored and the callback function is responsible for copying the appropriate fields from the child record buffer to the parent record buffer (and vice-versa).

The ctrouter.exe utility should be used to create a field mapping resource to the child table. Once the field mapping resource is set, and after the table is open by calling ctdbOpenTable() function, the two callback type mentioned above can be set to provide custom field mapping to and from the child table. It is recommended to set both CTDB_ON_RECORD_MAPTOPARENT and CTDB_ON_RECORD_MAPTOCHILD using a table handle, as this will automatically set the callbacks for any records allocated using the table handle. Example:

```c
/* open the child table */
if ((Retval = ctdbOpenTable(hTable, "child", CTOPEN_NORMAL)) == CTDBRET_OK)
{
    ctdbSetCallback(hTable, CTDB_ON_RECORD_MAPTOPARENT, MapToParent);
    ctdbSetCallback(hTable, CTDB_ON_RECORD_MAPTOCHILD, MapToChild);
}
```

CTDB_ON_RECORD_MAPTOCHILD

Called only if the child table has the field mapping resource activated. Overrides the field mapping information, that is, if the callback function pointers are set, then the field mapping information stored in the child table resource is ignored and the callback function is responsible for copying the appropriate fields from the child record buffer to the parent record buffer (and vice-versa).

The ctrouter.exe utility should be used to create a field mapping resource to the child table. Once the field mapping resource is set, and after the table is open by calling ctdbOpenTable() function, the two callback type mentioned above can be set to provide custom field mapping to and from the child table. It is recommended to set both CTDB_ON_RECORD_MAPTOPARENT and CTDB_ON_RECORD_MAPTOCHILD using a table handle, as this will automatically set the callbacks for any records allocated using the table handle. Example:

```c
/* open the child table */
if ((Retval = ctdbOpenTable(hTable, "child", CTOPEN_NORMAL)) == CTDBRET_OK)
{
    ctdbSetCallback(hTable, CTDB_ON_RECORD_MAPTOPARENT, MapToParent);
```
ctdbSetCallback(hTable, CTDB_ON_RECORD_MAPTOCHILD, MapToChild);
}

## Working with Callbacks

You must register your callback functions before they are invoked by c-treeDB code. The callback function is registered with a call to the `ctdbSetCallback()` function, passing the appropriate c-treeDB handle, the callback function type and the address of a function to receive the callback calls.

```c
CTDBRET ctdbSetCallback(CTHANDLE Handle, CTDB_CALLBACK_TYPE CallBackType,
                         ctdbCallbackFunction CallBackFunc);
```

### c-treeDB C API Example

/* allocate a new session handle */
CTHANDLE hSession = ctdbAllocSession(CTSESSION_CTREE);

/* set table open callback */
if (ctdbSetCallback(hSession, CTDB_ON_TABLE_OPEN, OnTableOpen) != CTDBRET_OK)
    printf("ctdbSetCallback failed\n");

You can register any of the defined callback functions using the session handle and every time a database, table or record handle is allocated, they will automatically inherit their callbacks from the session handle. Conversely, if you register callbacks with a database handle, every time a table or a record handle is allocated they will automatically inherit their callbacks from the database handle. Record handles will inherit any callbacks registered with the table handle.

You clear callbacks from a handle by calling either `ctdbClearCallback()` or `ctdbClearAllCallback()`.

### c-treeDB C API Example

/* allocate a record handle */
CTHANDLE hRecord = ctdbAllocRecord(hTable);

/* make sure there are no callbacks */
ctdbClearAllCallback(hRecord);

You can check if a given callback has been registered with a session, database, table or record handle by calling the `ctdbGetCallback()` function. If a callback function was set, `ctdbGetCallback()` returns the address of the function. If a particular callback is not set, `ctdbGetCallback()` returns NULL.

### c-treeDB C API Example

/* allocate a table handle */
CTHANDLE hTable = ctdbAllocTable(hDatabase);

/* make sure CTDB_ON_TABLE_OPEN callback is set */
if (ctdbGetCallback(hTable, CTDB_ON_TABLE_OPEN) == NULL)
    if (ctdbSetCallback(hTable, CTDB_ON_TABLE_OPEN, OnTableOpen) != CTDBRET_OK)
      printf("ctdbSetCallback failed\n");
Allocating and Freeing Memory Inside Callbacks

When a callback function is executed, it may need to allocate, re-allocate and release memory that was allocated by the c-treeDB internal code. The callback function must use exactly the same memory allocation function that was used by the c-treeDB code or a heap corruption, and most certainly a memory exception, will occur.

The c-treeDB session handle has a function pointer called onAlloc that must be used by the callback functions to allocate memory. The onAlloc function pointer has the following type:

```c
typedef pVOID (ctdbDECL* ctdbAllocFunc)(VRLEN size);
```

`size` is the number of bytes to be allocated. The returned value is a pointer to void.

The c-treeDB session handle also has a function pointer called onFree that must be used by callback function to release memory. The onFree function pointer has the following type:

```c
typedef void (ctdbDECL* ctdbFreeFunc)(pVOID ptr);
```

`ptr` points to memory to be released. No value is returned.

Please note that the c-treeDB database, table and record handles have a member variable called pSession which points to the current session handle. You can use this pSession member variable to obtain the reference to the onAlloc and onFree function pointers. In some cases you should typecast the pSession member as a pointer to a CTDBSESSION structure.
3.10 Working with Unicode

c-treeDB provides support for Unicode. This support includes:

- Unicode UTF-16 field types
- UTF-8 compliant C/C++ API
- Indexing on Unicode field data
- ICU library support

**c-treeDB Support for Unicode**

c-treeDB provides support for Unicode. This support includes:

- Unicode UTF-16 field types
- UTF-8 compliant C/C++ API
- Indexing on Unicode field data
- ICU library support

**Unicode UTF-16**

UCS-2 and UTF-16 are alternative names for a 16-bit Unicode Transformation Format, a character encoding form that provides a way to represent a series of abstract characters from Unicode and ISO/IEC 10646 as a series of 16-bit words suitable for storage or transmission via data networks. UTF-16 is officially defined in Annex Q of ISO/IEC 10646-1. It is also described in "The Unicode Standard" version 3.0 and higher, as well as in the IETF’s RFC 2871.

UTF-16 represents a character that has been assigned within the lower 65536 code points of Unicode or ISO/IEC 10646 as a single code value equivalent to the character’s code point: 0 for 0, hexadecimal FFFD for FFFD, for example.

UTF-16 represents a character above hexadecimal FFFF as a surrogate pair of code values from the range D800-DFFF. For example, the character at code point hexadecimal 10000 becomes the code value sequence D800 DC00, and the character at hexadecimal 10FFFF, the upper limit of Unicode, becomes the code value sequence DBFF DFFD. Unicode and ISO/IEC 10646 do not assign characters to any of the code points in the D800-DFFF range, so an individual code value from a surrogate pair does not ever represent a character.

These code values are then serialized as 16-bit words, one word per code value. Because the endian-ness of these words varies according to the computer architecture, UTF-16 specifies three encoding schemes: UTF-16, UTF-16LE, and UTF-16BE.

UTF-16 is the native internal representation of text in the NT/2000/XP versions of Windows and in the Java and .NET bytecode environments, as well as in Mac OS X’s Cocoa and Core Foundation frameworks.
**Unicode UTF-8**

UTF-8 is the byte-oriented encoding form of Unicode. The UTF-8 encoding is defined in ISO 10646-1:2000 Annex D and also described in RFC 3629 as well as section 3.9 of the Unicode 4.0 standard.

UTF-8 has the following properties:

- UCS characters U+0000 to U+007F (ASCII) are encoded simply as bytes 0x00 to 0x7F (ASCII compatibility). This means that files and strings which contain only 7-bit ASCII characters have the same encoding under both ASCII and UTF-8.
- All UCS characters >U+007F are encoded as a sequence of several bytes, each of which has the most significant bit set. Therefore, no ASCII byte (0x00-0x7F) can appear as part of any other character.
- The first byte of a multi-byte sequence that represents a non-ASCII character is always in the range 0xC0 to 0xFD and it indicates how many bytes follow for this character. All further bytes in a multi-byte sequence are in the range 0x80 to 0xBF. This allows easy re-synchronization and makes the encoding stateless and robust against missing bytes.
- All possible 231 UCS codes can be encoded.
- UTF-8 encoded characters may theoretically be up to six bytes long, however 16-bit BMP characters are only up to three bytes long.
- The sorting order of big endian UCS-4 byte strings is preserved.
- The bytes 0xFE and 0xFF are never used in the UTF-8 encoding.

**c-treeDB C++ API UTF-8 Compliance**

The support routines were changed to enable c-treeDB to accept Unicode UTF-8 strings for names of objects such as databases, tables, fields, indices, etc., and for path names used by the API.

For example, a table with name canção (a "song" in Portuguese) can be created with the following code.

**Example**

```c
TEXT tableName[9] = {0x63, 0x61, 0x6e, 0xc3, 0xa7, 0xc3, 0xa3, 0x6f, 0x00};
cTable->Create(tableName, CTCREATE_NORMAL);
```

The bytes assigned to variable `tableName` is the UTF-8 representation of the Portuguese word canção.

If the original strings are encoded using UTF-16, you can use the c-treeDB function, `ctdb_u16TOu8()`, to convert a UTF-16 string to a UTF-8 encoding. A UTF-8 string can also be converted back to UTF-16 by calling function `ctdb_u8TOu16()`.

**Example**

```c
CTTable* OpenTable(pWCHAR tableName)
{
    CTTable *ctTable = new CTTable(ctDB);
    CTString name(tableName);
    ctTable->Open(name, CTOPEN_NORMAL);
    Return ctTable;
}
```
Activating c-treeDB Unicode support

Unicode support is currently available for any client when connecting to the c-tree Server for Windows or Mac OS X and for c-treeACE Standalone libraries under Windows or Mac OS X. For client operation, ensure you install the c-tree Server for Windows or Mac OS X WITH Unicode support.

When building the c-treeACE libraries, execute mtmake with the "u" flag

```
mtmake u
```

to prepare the library for Unicode support. Standalone and client builds need the ICU libraries from the ICU web site, as described in next chapter.

c-treeDB C and C++ Unicode support is activated by defining macro ctdbUNICODE. ctdbUNICODE is activated automatically when ctUNICODE is selected with the mtmake build utility:

```
---- FairCom c-tree Plus UniCode Support ----
This version of c-tree Plus provides support for UNICODE field types.
```

Do you want to support UniCode field types? (Y)es (N)o (D)efaults- [N]: y

ICU - International Components for Unicode

Unicode is the single, universal character set for text which enables the interchange, processing, storage and display of text in many languages.

The International Components for Unicode (ICU) are a mature, widely used set of C/C++ and Java libraries providing Unicode and Globalization support for software applications. ICU is widely portable and gives applications the same results on all platforms and between C/C++ and Java software.

Complete details on the ICU can be found here: http://site.icu-project.org/
(http://site.icu-project.org/)

ICU is released under a nonrestrictive open source license that is suitable for use with both commercial software and with other open source or free software. To the extent required by the licenses accompanying the ICU libraries, the terms of such license will apply in lieu of the terms of any agreement with FairCom with respect to the open source software including, without limitation, any provisions governing access to source code, modification or reverse engineering. FairCom makes no representation, warranty or other commitment of any kind regarding such open source software, offers no technical support for such open source software and shall, to the maximum extent permitted by law, have no liability associated with its use.

Unicode Support

Simply storing Unicode data has always been possible with c-treeDB, provided the application treated the data as binary and performed any necessary translations. In this case using the stored Unicode data as a segment of an index was difficult since c-treeDB had no way of knowing how the underlying binary data was encoded: UTF-8. UTF-16, ASCII, etc.
c-treeACE Unicode UTF-16 Field Types

Storing Unicode data requires DODA entries for each field. The individual wide-characters used in UTF16 are not platform independent with respect to byte ordering. They are treated the same as short integers: on LOW_HIGH platforms, the lower order byte comes before the higher order byte. With the DODA entries in place, the Server and clients manage byte-order translation automatically.

C-treeDB has four Unicode UTF-16 field types:

<table>
<thead>
<tr>
<th>UTF-16 Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT_FUNICODE</td>
<td>A fixed length field containing a UTF-16 encoded, null terminated string. This Unicode field type is similar to CT_FSTRING field type.</td>
</tr>
<tr>
<td>CT_F2UNICODE</td>
<td>A fixed length field that begins with a 2-byte (16 bit) integer specifying the number of bytes in the following UTF-16 encoded string. This Unicode field type is similar to CT_F2STRING field type.</td>
</tr>
<tr>
<td>CT_UNICODE</td>
<td>A variable length field containing a UTF-16 encoded, null terminated string. This Unicode field type is similar to CT_STRING field type.</td>
</tr>
<tr>
<td>CT_2UNICODE</td>
<td>A variable length field that begins with a 2 byte (16 bit) integer specifying the number of bytes in the UTF-16 encoded string. This Unicode field type is similar to CT_2STRING field type.</td>
</tr>
</tbody>
</table>

The length fields at the beginning of CT_F2UNICODE and CT_2UNICODE field types, and the length in the DODA entry for CT_FSTRING and CT_F2STRING field types, are specified in bytes. Specifying a field length in bytes is consistent with all other c-treeDB field types, but is inconsistent with the system level routines that ordinarily use a number of characters, not a number of bytes, to describe the length of UTF-16 strings.

Storing a UTF-16 string longer than 64Kbytes requires a CT_UNICODE field. To store a UTF-16 string greater than 64Kbytes with a length prefix, convert the string to UTF-8 and store it in a CT_4STRING field, as discussed below. If this UTF-8 converted field is to be part of a key segment, then "Extended Key Segment" information must also be added to this key segment.

Creating Tables with Unicode Field types

Creating tables with Unicode field types is done by adding or inserting a new field with a field type set to one of the following Unicode field types: CT_FSTRING, CT_F2STRING, CT_STRING or CT_2STRING.

C-treeDB C++ API Example

```cpp
hTable.AddField("f1", CT_FUNICODE, 40);
hTable.AddField("f2", CT_INT4, 4);
hTable.Create("table", CTCREATE_NORMAL);
```
Reading UTF-16 Field Data

Two methods, **GetFieldAsUTF16()**, were added to the CTRecord class to enable applications to read data from a Unicode field types.

**By Field Number**

**GetFieldAsUTF16()** retrieves the field data as a Unicode UTF-16 string. If the underlying field type is not one of the Unicode field types, the data is converted to a UTF-16 string. **FieldName** is the number of the field, **value** is a pointer to a wide (UTF-16) string buffer and **size** indicates the size in bytes of the string area. Declaration:

```c
void CTRecord::GetFieldAsUTF16(NINT FieldNumber, pWCHAR value, VRLEN size);
```

**By Field Name**

**GetFieldAsUTF16()** retrieves the field data as Unicode UTF-16 string. If the underlying field type is not one of the Unicode field types, the data is converted to a UTF-16 string. **FieldName** is a string object representing the field name, **value** is a pointer to a wide (UTF-16) string buffer and **size** indicates the size in bytes of the string area. Declaration:

```c
void CTRecord::GetFieldAsUTF16(const CTString& FieldName, pWCHAR value, VRLEN size);
```

Reading UTF-16 Data C++ Example

```c
void CheckData(CTRecord& hRecord, const CTString& str, NINT val)
{
    WCHAR WStr[32];
    TEXT s[64];
    CTsigned t;

    hRecord.GetFieldAsUTF16(0, WStr, sizeof(WStr));
    ctdb_u16TOu8(WStr, s, sizeof(s));
    if (strcmp(s, str) != 0)
        printf("UNICODE field contents not the same written");

    t = hRecord.GetFieldAsSigned(1);
    if ((NINT)t != val)
        printf("integer field contents not the same written");
}
```

Writing UTF-16 Field Data

A new function **ctdbSetFieldAsUTF16()** has been added to c-treeDB C API to enable applications to write data to Unicode field types.

```c
CTDBRET ctdbSetFieldAsUTF16(CTHANDLE Handle, NINT FieldNbr, pWCHAR pValue);
```

**ctdbSetFieldAsUTF16()** puts a Unicode UTF-16 string in a Unicode field. If the underlying field type is not one of the Unicode field types, the UTF-16 string is converted to the appropriate type before the data is stored in the field. **Handle** is a record handle, **FieldName** is the field number and **pValue** is a pointer to a wide (UTF-16) string buffer. **ctdbSetFieldAsUTF16()** returns **CTDBRET_OK** on success.

Two new methods, **SetFieldAsUTF16()**, have been added to the CTRecord class to enable applications to write data to Unicode field types.
void CTRecord::SetFieldAsUTF16(NINT FieldNumber, pWCHAR value);

**SetFieldAsUTF16()** puts a Unicode UTF-16 string in a Unicode field. If the underlying field type is not one of the Unicode field types, the UTF-16 string is converted to the appropriate type before the data is stored in the field. *FieldNbr* is a number representing the field number and *value* is the wide (UTF-16) string buffer.

void CTRecord::SetFieldAsUTF16(const CTString& FieldName, pWCHAR value);

**SetFieldAsUTF16()** puts a Unicode UTF-16 string in a Unicode field. If the underlying field type is not one of the Unicode field types, the UTF-16 string is converted to the appropriate type before the data is stored in the field. *FieldName* is the field name and *value* is the wide (UTF-16) string buffer.

### Writing UTF-16 Data C++ Example

```cpp
void AddData(CTRecord& hRecord, const CTString& str, NINT val)
{
    WCHAR WStr[32];

    hRecord.Clear();
    ctdb_u8ToU16(str, WStr, sizeof(WStr));
    hRecord.SetFieldAsUTF16(0, WStr);
    hRecord.SetFieldAsSigned(1, (CTSIGNED)val);
    hRecord.Write();
}
```

### Creating Key Segments based on Unicode Fields

Unicode key segments provide a challenge for two reasons:

- Unlike all other key segments previously implemented, the number of bytes stored in the key and the number of bytes of source data used to construct the key are not the same.
- The derivation of the binary sort key (segment) stored in the index from the source data is not a simple transformation.

To accommodate both of these challenges, c-tree Plus incorporated “extended key segments.” The concept of an extended key segment can be applied to virtually any non-standard key segment. Our first implementation is for Unicode keys.

Because of the complexity of the Unicode collation algorithm, and because of the incredible breadth of language and country support envisaged by Unicode, FairCom has chosen to implement Unicode key segments using the International Components for Unicode (ICU) open-source development project. The ICU implementation of Unicode support is available on a wide variety of platforms, but not every platform. The ICU web site can be accessed at:

IBM International Components for Unicode (ICU) (http://site.icu-project.org/)

### How to Specify a Unicode Key Segment

An ordinary c-treeDB key segment is defined by a field handle and mode. c-treeDB also allows the specification of a key segment using an offset, length and mode.
In the following example, since the segment mode is CTSEG_SCHSEG, and if hField is a handle of a field whose type is one of the Unicode field types CT_FUNICODE, CT_F2UNICODE, CT_UNICODE or CT_2UNICODE, then c-treeDB will understand this is a Unicode key segment.

Specifying a Unicode Key Segment C++ Example

```cpp
hField = hTable.AddField( "customer", CT_FUNICODE, 40);

hTable.AddSegment(0, hField, CTSEG_SCHSEG);
```

Specifying a Unicode Key Segment with CTSEG_UNCSEG

If a key segment is a Unicode segment, but the segment mode is not one of the CTSEG_SCHSEG modes or the segment field is not one of the Unicode field types, the segment mode must also specify the Unicode segment modifier CTSEG_UNCSEG. For example, assume the CT_FSTRING field contains a UTF-8 string:

Notice an Extended Key Segment definition must be created for the segment. Please refer to section Extended Key Segment Definition (page 216) below. If no extended key segment definition is provided at the time of the table creation, c-treeDB will create an extended key segment with default values. Please refer to the Default Extended Key Segment Definition section below.

CTSEG_UNCSEG C++ API Example

```cpp
hField = hTable.AddField( "customer", CT_FSTRING, 40);

hTable.AddSegment(0, hField, CTSEG_SCHSEG | CTSEG_UNCSEG);
```

ICU Collation Option Overview

The collation options can be grouped as follows: locale default control, collation strength, normalization, and special attributes. Locale default control effects the degree to which a default locale must be related to the requested locale. Collation strength determines how case, accents and other character modifiers affect the ordering of sort keys. Normalization effects how alternative variations of the "same" character (including its accents and other modifiers) are compared. The special attributes effect particular properties of the collation, which further modify the strength and normalization options. For example, a special attribute can be used to force lower case characters to be first or last in the collation.

If no locale default control option is made part of kseg_comp, there is no restriction on how close to the requested locale the effective locale must be. For example, if you request collation for the German language ("de"), you are likely to get a locale based on the system default (e.g., "en_US" in the United States). This is not a problem since it has been determined that the default rules work for the German language.

If ctKSEG_COMPU_SYSDEFAULT_NOTOK is used, then a request to use locale "xx_YY_Variant" will succeed as long as collation rules for "xx" are available. If ctKSEG_COMPU_FALLBACK_NOTOK is used, then rules for the particular locale with its optional country and variant modifiers must be available. Falling back from "xx_YY" to "xx" is not satisfactory. In the case of the "de" locale noted above, the segment definition would cause an error in the call to PutXtdKeySegmentDef() if either of the "NOTOK" default restrictions are part of the definition.
At most one of the following collation strength options can be included in `kseg_comp`:

- `ctKSEG_COMPU_S_PRIMARY`
- `ctKSEG_COMPU_S_SECONDARY`
- `ctKSEG_COMPU_S_TERTIARY`
- `ctKSEG_COMPU_S_QUATERNARY`
- `ctKSEG_COMPU_S_IDENTICAL`
- `ctKSEG_COMPU_S_DEFAULT`

At most, one of the following normalization options can be included in `kseg_comp`:

- `ctKSEG_COMPU_N_NONE`
- `ctKSEG_COMPU_N_CAN_DECMP`
- `ctKSEG_COMPU_N_CMP_DECMP`
- `ctKSEG_COMPU_N_CAN_DECMP_CMP`
- `ctKSEG_COMPU_N_CMP_DECMP_CAN`
- `ctKSEG_COMPU_N_DEFAULT`

One or more of the following special attributes can be included in `kseg_comp` After each one of the c-tree symbolic constants is the equivalent ICU-attribute value pair.

<table>
<thead>
<tr>
<th>c-tree Symbolic Constant</th>
<th>ICU Attribute value pair</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ctKSEG_COMPU_A_FRENCH_ON</code></td>
<td><code>(UCOL_FRENCH_COLLATION, UCOL_ON)</code></td>
</tr>
<tr>
<td><code>ctKSEG_COMPU_A_FRENCH_OFF</code></td>
<td><code>(UCOL_FRENCH_COLLATION, UCOL_OFF)</code></td>
</tr>
<tr>
<td><code>ctKSEG_COMPU_A_CASE_ON</code></td>
<td><code>(UCOL_CASE_LEVEL, UCOL_ON)</code></td>
</tr>
<tr>
<td><code>ctKSEG_COMPU_A_CASE_OFF</code></td>
<td><code>(UCOL_CASE_LEVEL, UCOL_OFF)</code></td>
</tr>
<tr>
<td><code>ctKSEG_COMPU_A_DECOMP_ON</code></td>
<td><code>(UCOL_DECOMPOSITION_MODE, UCOL_ON)</code></td>
</tr>
<tr>
<td><code>ctKSEG_COMPU_A_DECOMP_OFF</code></td>
<td><code>(UCOL_DECOMPOSITION_MODE, UCOL_OFF)</code></td>
</tr>
<tr>
<td><code>ctKSEG_COMPU_A_SHIFTED</code></td>
<td><code>(UCOL_ALTERNATE_HANDLING, UCOL_SHIFTED)</code></td>
</tr>
<tr>
<td><code>ctKSEG_COMPU_A_NONIGNR</code></td>
<td><code>(UCOL_ALTERNATE_HANDLING, UCOL_NON_IGNORABLE)</code></td>
</tr>
<tr>
<td><code>ctKSEG_COMPU_A_LOWER</code></td>
<td><code>(UCOL_CASE_FIRST, UCOL_LOWER_FIRST)</code></td>
</tr>
<tr>
<td><code>ctKSEG_COMPU_A_UPPER</code></td>
<td><code>(UCOL_CASE_FIRST, UCOL_UPPER_FIRST)</code></td>
</tr>
<tr>
<td><code>ctKSEG_COMPU_A_HANGUL</code></td>
<td><code>(UCOL_NORMALIZATION_Mode, UCOL_ON WITHOUT HANGUL)</code></td>
</tr>
</tbody>
</table>

It is permissible to set `kseg_comp` to zero. A zero `kseg_comp` implies no restrictions on locale defaults, default collation strength, default normalization, and no special attributes.

For a complete treatment of all of these options, please refer to the ICU web site and the Unicode Consortium’s web site and publications.

**Storing UTF-8 Data**

Since a UTF8 encoded string is comprised of ordinary ASCII characters (with code values between 0 and 127) and multi-byte characters (which have the highest-order bit set in each byte),
they can be stored normally in any of c-treeDB’s string or binary field types such as CT_STRING, CT_FSTRING, CT_4STRING, etc. It is up to the application to decipher the field data.

On the other hand, if a field holding a UTF-8 encoded string is part of a key segment, then you need to define an “Extended Key Segment” for that segment to allow c-treeDB to apply the appropriate translations to the field data when building the key data. Please refer to the “Extended Key Segment Definition” section above for more details.

**Converting from Unicode UTF-16 to UTF-8**

c-treeACE provides conversion routines between UTF8 and UTF16. The input strings are assumed to be terminated by a NULL character. All output buffer sizes are specified in bytes. The conversion routines return CTDBRET_OK (0) on success, error VBSZ_ERR (153) if the output buffer is too small, or error BMOD_ERR (446) if there is a problem with the input string.

`ctdb_u8TOu16()` converts an ASCII or UTF-8 encoded string to a UTF-16 Unicode string:

```c
NINT ctdb_u8TOu16(pTEXT u8str, pWCHAR u16str, NINT u16size);
```

**c-treeDB C API Example**

```c
WCHAR buffer[256];
switch (ctdb_u8TOu16("tablename", buffer, sizeof(buffer)))
{
    case CTDBRET_OK:
    {
        printf("UTF-8 to UTF-16 conversion ok\n");
        break;
    }
    case VBSZ_ERR:
    {
        printf("Conversion buffer is too small\n");
        break;
    }
    case BMOD_ERR:
    {
        printf("Problem occurred during conversion\n");
        break;
    }
    default:
    {
        printf("Unknown error code\n");
        break;
    }
}
```

`ctdb_u16TOu8()` converts a UTF-16 encoded string to a UTF-8 Unicode string:

```c
NINT ctu16TOu8(pWCHAR u16str, pTEXT u8str, NINT u8size);
```

**c-treeDB C API Example**

```c
TEXT buffer[512];
switch (ctdb_u16TOu8(tableName, buffer, sizeof(buffer)))
{
    case CTDBRET_OK:
    {
```
printf("UTF-16 to UTF-8 conversion ok\n");
break;
}
case VBSZ_ERR:
{
    printf("Conversion buffer is too small\n");
    break;
}
case BMOD_ERR:
{
    printf("Problem occurred during conversion\n");
    break;
}
default:
{
    printf("Unknown error code\n");
    break;
}
}

Extended Key Segment Definition

The implementation of extended key segments in c-treeDB allows a single extended key segment definition to be used by more than one actual key segment. Extended key segment definitions may be set for all segments of a table, all segments of an index or for each particular key segment.

If a key segment mode includes a modifier for an extended key segment definition CTSEG_UNCSEG or the segment mode is CTSEG_SCHSEG, and the field type is one of the Unicode types, then the particular extended key segment definition to use for this segment is determined according to the following hierarchy. Use the definition specified for:

1. The segment
2. The index associated with the segment
3. The data file associated with the index

Once an extended key segment definition has been specified at a particular level (for a particular type of segment), an attempt to specify another definition at the same level results in an error. This is in part because of the "first use" strategy noted above, and because one should not change a definition if key values already exist.
c-treeDB C++ API Methods

The following methods are available to implement table wide extended key segment definitions:

```cpp
class CTTable
{
public:
    void SetTableKSeg(pctKSEGDEF pKSeg);
    CTTable::SetTableKSeg(pctKSEGDEF pKSeg)
        establishes a table-wide extended key segment definition. pKSeg is a
        pointer to an extended key segment definition structure with the extended
        key definition.
    void GetTableKSeg(pctKSEGDEF pKSeg);
    CTTable::GetTableKSeg() retrieves the current table-wide extended key segment
        definition. pKSeg is a pointer to an extended key segment definition
        structure which will receive the definition.

doctor

The following two methods were added to implement index-wide extended key segment
definitions:

```cpp
class CTIndex
{
public:
    void SetIndexKSeg(pctKSEGDEF pKSeg);
    CTIndex::SetIndexKSeg(pctKSEGDEF pKSeg)
        establishes an index-wide extended key segment definition. pKSeg is
        a pointer to an extended key segment definition structure with the extended
        key definition.
    void GetIndexKSeg(pctKSEGDEF pKSeg);
    CTIndex::GetIndexKSeg() retrieves the current index-wide extended key segment
        definition. pKSeg is a pointer to an extended key segment definition
        structure which will receive the definition.

doctor

The following three methods were added to implement extended key segment definition for a
specific key segment:

```cpp
class CTSegment
{
public:
    void SetSegmentKSeg(pctKSEGDEF pKSeg);
    CTSegment::SetSegmentKSeg(pctKSEGDEF pKSeg)
        establishes a segment’s extended key segment definition. pKSeg is
        a pointer to an extended key segment definition structure with the extended
        key definition.
    void GetSegmentKSeg(pctKSEGDEF pKSeg);
    CTSegment::GetSegmentKSeg() retrieves the current index wide extended key segment
        definition. pKSeg is a pointer to an extended key segment definition
        structure which will receive the definition.
    void SetKSegDefaults(pctKSEGDEF pKSeg);
    CTSegment::SetKSegDefaults() sets the system-wide default values for the extended key
        segment definition. pKSeg is a pointer to an extended key segment definition
        structure which will receive the definition.

doctor

The default values are:

```cpp
kseg_sziz = ctKSEG_SSIZ_COMPUTED;
kseg_type = ctKSEG_TYPE_UNICODE;
kseg_styp = ctKSEG_STYP_UTF16;
kseg_comp = ctKSEG_COMPU_S_DEFAULT | ctKSEG_COMPU_N_NONE;
kseg_desc = "en_US"
```

Extended Key Segment Structure

Extended key segments are specified by filling the fields of the ctKSEGDEF structure:

```cpp
#define ctKSEGDEFLEN 32 /* length of desc string */
```
typedef struct keysegdef {
  LONG kseg_stat;    /* status (internal use) * /
  LONG kseg_vrsn;   /* version info */
  LONG kseg_ssiz;   /* source size */
  LONG kseg_type;   /* segment type */
  LONG kseg_styp;   /* source type */
  LONG kseg_comp;   /* comparison options */
  LONG kseg_rsv1;   /* future use */
  LONG kseg_rsv2;   /* future use */
  TEXT kseg_desc[ctKSEGDL]; /* text specification eg, locale string */
} ctKSEGDEF, ctMEM* pctKSEGDEF;

The c-treeACE module cport.h contains defines for all of the constants, beginning with ctKSEG, used to create an extended key segment definition. As extended key segments are currently implemented, the kseg_stat and the kseg_vrsn members are filled-in as needed by the extended key segment implementation itself. The kseg_ssiz member specifies the number of bytes of source data to use to derive the actual key segment. In addition to using a specific numeric value for the source size, kseg_ssiz may also be assigned either of two values discussed in the following two sections.

ctKSEG_SSIZ_COMPUTED

The information about the underlying data field will be used to compute how much source data is available. For fields without length specifiers (such as CT_STRING or CT_UNICODE) an appropriate version of strlen() will be used to determine data availability. However, this could be very inefficient if the field may hold very long strings since it is likely that only a small portion of the variable length field will actually contribute to the key segment. An alternative is to specify a fixed source size. If the variable data has less than this size, it will still be handled correctly.

ctKSEG_SSIZ PROVIDED

The call to create the key segment will provide the particular length of source data available.

For an ICU Unicode definition, the remaining structure members are specified as follows:

kseg_type

Must be set to ctKSEG_TYPE_UNICODE.

kseg_styp

Specify the type of source data as follows:

- ctKSEG_STYP_UTF8
- ctKSEG_STYP_UTF16
- ctKSEG_STYP_PROVIDED

ctKSEG_STYP_PROVIDED means that the type of source data will be determined at run-time during key value construction. (Key value construction consists of one or both of assembling the key value from its component segments and performing transformations to generate a binary sort key). In this case, if the data type is one of the conventional c-tree Plus string types (e.g., CT_STRING), the source data type is UTF8; if a Unicode string type is found (e.g., CT_UNICODE), then the source data type is UTF16. However, if the underlying data type does not fall into either of these categories, the data is treated as UTF16, and used as is.

kseg_desc
Contains the ICU locale formed as an ordinary, null-terminated ASCII string. The format specified by ICU is "xx", "xx_YY", or "xx_YY_Variant" where "xx" is the language as specified by ISO-639 (e.g., "fr" for French); "YY" is a country as specified by ISO-3166 (e.g., "fr_CA" for French language in Canada); and the "Variant" portion represents system-dependent options. Note: When ICU uses a locale to access collation rules, it attempts to get rules for the closest match to the locale specified in *kseg_desc*. By default, there is no restriction on how close the match of locales must be to be acceptable. You can restrict the use of alternative locales by including either *ctKSEG_COMPU_FALLBACK_NOTOK* or *ctKSEG_COMPU_SYSDEFAULT_NOTOK* as part of the bit map comprising kseg_comp discussed below. After a successful call to *PutXtdKeySegmentDef()* the *GetXtdKeySegmentDef()* function can be used to determine the actual ICU locale used during collation.

**kseg_comp**

This member of the structure permits the full range of ICU collation options to be specified through a bit map.

**Example of extended key segment structure:**

```c
ctKSEGDEF ksgdef;
ksgdef.kseg_ssz = 12; /* 12 bytes for the source */
ksgdef.kseg_type = ctKSEG_TYPE_UNICODE; /* ICU Unicode */
ksgdef.kseg_styp = ctKSEG_STYP_UTF16; /* UTF16 source data */
ksgdef.kseg_comp = ctKSEG_COMPU_A_LOWER; /* lower case sorts first */
strcpy(ksgdef.kseg_desc,"fr_CA"); /* French in Canada */
```
3.11 Compatibility with other c-tree API Technologies

Compatibility with c-treeACE ISAM and Low-Level Data Files

The basic requirement for c-treeACE ISAM and low-level data and index files to work with c-treeDB is that the data file must have a DODA describing the fields and an IFIL describing the data and index structures. The DODA resource must have an entry for every field that forms a data record.

The c-treeDB interface was designed to work best with ISEG structures with schema segments, that is, each index segment is a field declared in a DODA entry. c-treeDB works with index segments defined in the ISEG structures using absolute record offsets and arbitrary lengths, that may or may not span more than one field in the record buffer, however, advanced features of c-treeDB may not work as expected.

.ctdbAlterTable(), .ctdbSetRecordOn(), and .ctdbFindRecord() are typical examples of functions that may not work as expected when dealing with absolute segment offsets.

Another source of problems may come from non-standard usage of c-tree field types declared in the DODA. There have been cases where CT_ARRAY fields being used as variable length data, CT_STRING fields being used as fixed string fields. Another source of concern are CT_DATE, CT_TIME, and CT_DATETIME fields whose contents will not be compatible with the definitions of date and time used by c-treeDB.

c-treeDB was designed from the outset to be as compatible as possible with existing c-tree ISAM and low-level data, however, there are situations where c-treeDB may not be able to handle very specific data type arrangements used by c-tree Plus users in their custom applications.

Compatibility with c-treeACE SQL

Tables created with c-treeDB are 100% compatible with c-treeACE SQL, even when tables are created without RECBYT or ROWID indices. Tables created with c-treeDB but without transaction processing flags will not be able to participate in SQL transactions or transaction isolation levels.

Tables created by c-treeACE applications must be 100% compatible with c-treeDB to get the full benefit of all c-treeACE SQL commands and features. c-treeACE SQL makes extensive use of .ctdbAlterTable(), .ctdbFindRecord() and .ctdbRecordSetOn(); any c-treeACE tables that do not support these operations will have reduced compatibility with c-treeACE SQL.

Field mapping between c-treeACE SQL, c-treeDB and c-treeDB .NET

<table>
<thead>
<tr>
<th>c-treeACE SQL Field Type</th>
<th>c-treeDB/c-treeDB .NET Field Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>CT_BOOL</td>
</tr>
<tr>
<td>TINYINT</td>
<td>CT_TINYINT</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>CT_SMALLINT</td>
</tr>
<tr>
<td>INTEGER</td>
<td>CT_INTEGER</td>
</tr>
<tr>
<td>c-treeACE SQL Field Type</td>
<td>c-treeDB/c-treeDB .NET Field Type</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>FLOAT</td>
<td>CT_FLOAT</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>CT_DOUBLE</td>
</tr>
<tr>
<td>TID</td>
<td>CT_UINTeger</td>
</tr>
<tr>
<td>CHAR</td>
<td>CT_CHARS</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>CT_VARCHAR</td>
</tr>
<tr>
<td>LVARCHAR</td>
<td>CT_LVARCHAR</td>
</tr>
<tr>
<td>BINARY</td>
<td>CT_BINARY</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>CT_VARBINARY</td>
</tr>
<tr>
<td>LVARBINARY</td>
<td>CT_LVARBIN</td>
</tr>
<tr>
<td>DATE</td>
<td>CT_DATE</td>
</tr>
<tr>
<td>TIME</td>
<td>CT_TIME</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>CT_TIMESTAMP</td>
</tr>
<tr>
<td>BIGINT</td>
<td>CT_BIGINT</td>
</tr>
<tr>
<td>MONEY</td>
<td>CT_CURRENCY</td>
</tr>
<tr>
<td>NUMBER, NUMERIC, DECIMAL</td>
<td>CT_NUMBER</td>
</tr>
</tbody>
</table>
3.12 c-treeDB, ISAM, and Low-Level Integration

If you are creating a new application and have selected c-treeDB as the API of choice to access your data, there may be situations where you need to place calls directly into the ISAM or even low-level layers while remaining in your c-treeDB code. This may be to obtain certain specific services that are not directly supported by c-treeDB, or you may want to rewrite certain c-treeDB functionality to better suit your specific requirements.

It may also be common to find situations were you have an existing application written using the ISAM or low-level API, however, you develop new modules using the c-treeDB API and will migrate the existing modules over time to c-treeDB. In either case you will need c-treeDB to support the mix of ISAM or low-level function calls with your c-treeDB code, specifically when you must work with a table's data and index files and record data.

Functionality has been added to more easily support using c-treeDB with multiple APIs.

Overview

c-treeDB, short for c-tree DataBase, is a high-level, easy-to-use API abstracting the c-tree ISAM and low-level APIs. c-treeDB is intended as the standard for c-tree Plus programming.

The C functions and C++ methods described below present new functionality supplementing the c-treeDB C and C++ APIs.

Switching c-tree instances

If a c-treeDB application has multiple sessions, it may be necessary to force a c-tree instance switch before directing calls to ISAM and low-level functions to ensure those calls are made in the correct context.

This is particularly important in LOCLIB applications were you have one session connected to a c-tree Server, the remote session, and another session performing local I/O. In this case, it is very important to closely control which c-tree instance you require before making calls into the ISAM and low-level function layers.

Almost all c-treeDB functions automatically perform a c-tree instance switch for you; you need only take concern with c-tree instance switching in the case where you make ISAM or low-level calls within c-treeDB code.

The following c-treeDB C function performs a c-tree instance switch:

```c
CTDBRET ctdbSwitchInstance(CTHANDLE Handle)
```

This call will force a switch to the c-tree Plus instance indicated by the Session handle. Each session handle has a unique c-tree instance id. When most c-treeDB functions are called, they automatically perform a c-tree instance switch. `ctdbSwitchInstance` is used before a call to a specific c-tree ISAM or low level function to ensure the correct instance is active before instantiating the call. You may pass any c-treeDB handle to `ctdbSwitchInstance`. CTDBRET_OK is returned on success.

Similarly, the following c-treeDB C++ method performs a c-tree instance switch:
void CTBase::SwitchInstance()

This method will force a switch to the c-tree instance indicated by the Session object. Each session object has a unique c-tree instance id. When most c-treeDB C++ methods are called, they automatically perform a c-tree instance switch. If any errors are detected, a CTException is thrown.

The following is an example demonstrating a server administration logon in a LOCLIB implementation then forcing a c-tree instance switch to the remote instance and calling some ctreeUserOperation function.

/* declare and allocate the remote and local session handles */
CTHANDLE hRemote = ctdbAllocSession(CTSESSION_CTREE);
CTHANDLE hLocal = ctdbAllocSession(CTSESSION_CTREE);

/* logon to c-tree server using the remote session handle */
if (ctdbLogon(hRemove, "FAIRCOMS", "ADMIN", "ADMIN") != CTDBRET_OK)
    printf("Remote ctdbLogon failed\n");

/* logon to local session using the local session handle */
if (ctdbLogon(hLocal, "local", "ADMIN", "ADMIN") != CTDBRET_OK)
    printf("Local ctdbLogon failed\n");

/* perform a c-tree instance switch and call ctreeUserOperation function */
if (ctdbSwitchInstance(hRemote) != CTDBRET_OK)
    printf("ctdbSwitchInstance failed\n");
else
    CtreeUserOperation("!mkdir faircom", buffer, sizeof(buffer));

Switching ISAM contexts

Each time a record handle is allocated with ctdbAllocRecord, the allocated record handle acquires its own ISAM context, which means each record position operates independently from the other records. Record operations that move the current record position of one record handle will not interfere with other record handles.

If a c-treeDB application requires a call to the ISAM or low-level functions, it should ensure those calls are made in the correct ISAM context. All c-treeDB record handling functions automatically perform an ISAM context switch.

The following c-treeDB C function is used to perform a context switch:

CTDBRET ctdbSwitchContext(CTHANDLE Handle)

This call will force a switch to the c-tree ISAM context indicated by the record handle. Each record handle has a c-tree ISAM context id associated with it. When most c-treeDB record handling functions are called, they will automatically perform a c-tree ISAM context switch. ctdbSwitchContext is called before specific c-tree ISAM or low level calls to make sure the correct ISAM context is active before making those calls. The handle must be a record handle. No other handle is acceptable. CTDBRET_OK is returned on success.

Similarly, the following c-treeDB C++ method should be used to perform a context switch:

void CTreeRecord::SwitchContext()

This method will force a switch to the c-tree ISAM context indicated by the record object. Each record object may have its own c-tree ISAM context id. If any errors are detected, a CTException is thrown.
The following code snippet demonstrates use of the `ctdbSwitchContext` function to call the c-tree ISAM function `ResetRecord`.

/* force a context switch */
if (ctdbSwitchContext(hRecord) != CTDBRET_OK)
    printf("ctdbSwitchContext failed\n");

/* call ResetRecord */
if (ResetRecord((COUNT)ctdbGetDatno(hRecord), SWTCURI))
    printf("ResetRecord failed\n");

Obtaining table data and file number

Most c-tree ISAM and low-level functions require a data or index file number to operate correctly. Data file operations may require a data file number while all index operations will require an index file number.

The c-treeDB C and C++ APIs provide several functions and methods to extract the data and index file numbers from c-treeDB record or table handles.

Obtaining data file number

The following c-treeDB function will retrieve a data file number (or `datno`) from a table handle or any handle that can be converted into a table handle such as a record, segment, index and field handles:

```
NINT ctdbGetDatno(CTHANDLE Handle)
```

Retrieve the table `datno`. Handle must be a table handle, or a handle that can be converted into a table handle. Return the table `datno` on success or -1 on failure. If `ctdbGetDatno()` returns -1, the error code can be retrieved by calling the `ctdbGetError()` function.

The following c-treeDB method will similarly retrieve a data file number from a table object:

```
NINT CTTable::GetDatno()
```

If the `GetDatno()` method fails, a `CTException` is thrown.

An example using the c-treeDB C API:

```
CTDBRET DeleteTable(CTHANDLE hSession, pTEXT tablename)
{

    CTDBRET Retval = CTDBRET_OK;
    CHANDLE hTable = ctdbAllocTable(hSession);

    if (hTable)
    {
        /* open the table exclusive */
        if ((Retval = ctdbOpenTable(hTable, tablename, CTOPEN_EXCLUSIVE)) != CTDBRET_OK)
            return Retval;

        /* delete a file */
        if ((Retval = (CTDBRET)DeleteRFile((COUNT)ctdbGetDatno(hTable)) != CTDBRET_OK)
            return Retval;
    }
    else
    {
        Retval = CTDBRET_NOMEMORY;
        return Retval;
    }

```
Obtaining index file number

Three c-treeDB functions have been added to the c-treeDB API, which allow the retrieval of an index file number from a c-treeDB handle.

cmdbGetIdxno(), will retrieve an index file number from a c-treeDB handle and is declared as follows:

NINT cmdbGetIdxno(CTHANDLE Handle)

- Handle must be an index or segment handle.

cmdbGetIdxnoByName() will retrieve an index file number given an index name and is declared as follows:

NINT cmdbGetIdxnoByName(CTHANDLE Handle, pTEXT indexname)

- Handle must be a table handle, or a handle that can be converted into a table handle.
- IndexName is a string containing the index name.

To retrieve the index file number by index number, call the c-treeDB function cmdbGetIdxnoByNumber() declared as follows:

NINT cmdbGetIdxnoByNumber(CTHANDLE Handle, NINT index)

- Handle must be a table handle, or a handle that can be converted into a table handle.
- index is a c-treeDB index number. The first index number is zero.

These c-treeDB functions will return the index number on success or -1 on failure. If -1 is returned, the error code is retrieved with a call to the cmdbGetError() function.

Corresponding methods are available in the c-treeDB C++ API. The following method is used to retrieve the data file number from a CTTable object:

NINT CTTable::GetDatno()

This retrieves the table datno. A CException is thrown if an error occurs.

The following methods are used to retrieve the index file number:

NINT CTIndex::GetIdxno()

This method retrieves the index file number from the index object.

NINT CTTable::GetIdxno(const CTString& IndexName)

This method retrieves the index file number from the table object, given the index name.

NINT CTTable::GetIdxno(NINT index)

This method retrieves the index file number from the table object, given the c-treeDB index number.

In all cases, if the GetIdxno() method fails, a CException is thrown.

Below is a snippet demonstrating the c-treeDB C API function:

/* retrieve the first key of first index */
TEXT keyval[256];

if (cmdbGetIdxnoByNumber(hTable, 0), keyval)
    printf("FirstKey failed\n");
4. c-treeDB C++ API Class Reference

This chapter is a reference of the classes in the c-treeDB C++ API, which are listed below and described in more detail in the following sections.

- **CTBase**: c-treeDB Base class, used to make other classes.
- **CTBigint**: Big integer data type class
- **CTBlob**: Blob data type class
- **CTCurrency**: Currency data type class
- **CTDatabase**: Database management class
- **CTDate**: Date data type class
- **CTDateTime**: DateTime data type class
- **CTException**: Exception handler
- **CTField**: Field management class
- **CTIndex**: Index management class
- **CTNumber**: Number management class
- **CTMoney**: Money data type class
- **CTNumber**: Number management class
- **CTRecord**: Record management class
- **CTSegment**: Segment management class
- **CTSession**: Session management class
- **CTString**: String data type class
- **CTTable**: Table management class
- **CTTime**: Time data type class
4.1 c-treeDB Definitions

This section contains definitions of important c-treeDB elements.

Field Types

c-treeDB, c-treeDB .NET, and c-treeDB VCL support all original c-treeACE field types and includes redefinition for new field types. For compatibility reasons, the original c-tree Plus field types can be used, but FairCom suggests using the new field types.

Note that these do not represent new field types, however, substitute a new name of existing c-treeACE field types. The new naming convention is used within the c-treeACE SQL product line, and offers a better description of the fields.

There is absolutely no difference in using any of the definitions, however, for future compatibility and better program reading, the new field types offer an improved path.

<table>
<thead>
<tr>
<th>c-treeDB Field Type</th>
<th>c-treeACE Field Type</th>
<th>Equivalent Data Type</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT_BOOL</td>
<td>CT_BOOL</td>
<td>CTBOOL</td>
<td>One-byte Boolean</td>
</tr>
<tr>
<td>CT_TINYINT</td>
<td>CT_CHAR</td>
<td>CTSGIGNED</td>
<td>Signed one-byte integer.</td>
</tr>
<tr>
<td>CT_UTINYINT</td>
<td>CT_CHARU</td>
<td>CTUNSIGNED</td>
<td>Unsigned one-byte integer.</td>
</tr>
<tr>
<td>CT_SMALLINT</td>
<td>CT_INT2</td>
<td>CTSGIGNED</td>
<td>Signed two-byte integer.</td>
</tr>
<tr>
<td>CT_USMALLINT</td>
<td>CT_INT2U</td>
<td>CTUNSIGNED</td>
<td>Unsigned two-byte integer.</td>
</tr>
<tr>
<td>CT_INTEGER</td>
<td>CT_INT4</td>
<td>CTSGIGNED</td>
<td>Signed four-byte integer.</td>
</tr>
<tr>
<td>CT_UINTERGER</td>
<td>CT_INT4U</td>
<td>CTUNSIGNED</td>
<td>Unsigned four-byte integer.</td>
</tr>
<tr>
<td>CT_MONEY</td>
<td>CT_MONEY</td>
<td>CTMONEY</td>
<td>Signed four-byte integer interpreted as number of pennies (two fixed decimal places)</td>
</tr>
<tr>
<td>CT_DATE</td>
<td>CT_DATE</td>
<td>CTDATE</td>
<td>Unsigned four-byte integer interpreted as date.</td>
</tr>
<tr>
<td>CT_TIME</td>
<td>CT_TIME</td>
<td>CTTIME</td>
<td>Unsigned four-byte integer interpreted as time.</td>
</tr>
<tr>
<td>CT_FLOAT</td>
<td>CT_SFLOAT</td>
<td>CTFLOAT</td>
<td>Four-byte floating point.</td>
</tr>
<tr>
<td>CT_DOUBLE</td>
<td>CT_DFLOAT</td>
<td>CTFLOAT</td>
<td>Eight-byte floating point.</td>
</tr>
<tr>
<td>CT_TIMESTAMP</td>
<td>CT_TIMES</td>
<td>CTDATETIME</td>
<td>Time stamp.</td>
</tr>
<tr>
<td>CT_EFLOAT</td>
<td>CT_EFLOAT</td>
<td>CTFLOAT</td>
<td>Extended precision floating point (not supported as a key segment).</td>
</tr>
<tr>
<td>CT_BINARY</td>
<td>CT_F2STRING</td>
<td>pTEXT, pUTEXT</td>
<td>Arbitrary fixed length data. Fixed length binary data</td>
</tr>
<tr>
<td>CT_CHARS</td>
<td>CT_FSTRING</td>
<td>pTEXT</td>
<td>Fixed-length delimited data. Fixed-length string data</td>
</tr>
</tbody>
</table>
### c-treeDB Field Type vs c-treeACE Field Type

<table>
<thead>
<tr>
<th>c-treeDB Field Type</th>
<th>c-treeACE Field Type</th>
<th>Equivalent Data Type</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT_FPSTRING</td>
<td>CT_FPSTRING</td>
<td>pTEXT</td>
<td>Fixed-length data with 1-byte length count</td>
</tr>
<tr>
<td>CT_F2STRING</td>
<td>CT_F2STRING</td>
<td>pTEXT</td>
<td>Fixed-length data with 2-byte length count</td>
</tr>
<tr>
<td>CT_F4STRING</td>
<td>CT_F4STRING</td>
<td>pTEXT</td>
<td>Fixed-length data with 4-byte length count</td>
</tr>
<tr>
<td>CT_BIGINT</td>
<td>CT_BIGINT</td>
<td>CTBIGINT</td>
<td>Eight-byte signed integer</td>
</tr>
<tr>
<td>CT_NUMBER</td>
<td>CT_NUMBER</td>
<td>CTNUMBER</td>
<td>Scaled BCD number</td>
</tr>
<tr>
<td>CT_CURRENCY</td>
<td>CT_CURRENCY</td>
<td>CTCURRENCY</td>
<td>Eight-byte signed integer interpreted as currency value with four fixed decimal digits.</td>
</tr>
<tr>
<td>CT_PSTRING</td>
<td>CT_PSTRING</td>
<td>pTEXT</td>
<td>Varying length field data with 1-byte length count.</td>
</tr>
<tr>
<td>CT_VARBINARY</td>
<td>CT_2STRING</td>
<td>pTEXT</td>
<td>Varying length field data with 2-byte length count. Variable length binary data of up to 65535 bytes.</td>
</tr>
<tr>
<td>CT_LVB</td>
<td>CT_4STRING</td>
<td>pTEXT</td>
<td>Varying length field data with 4-byte length count. Variable length binary data of up to 4294967295 bytes.</td>
</tr>
<tr>
<td>CT_VARCHAR or CT_LVC</td>
<td>CT_STRING</td>
<td>pTEXT</td>
<td>Varying length field delimited data. Variable length string data.</td>
</tr>
<tr>
<td>CT_UBIGINT</td>
<td>CT_INT8U</td>
<td></td>
<td>8-byte unsigned integer.</td>
</tr>
<tr>
<td>CT_NCHAR</td>
<td>CT_F2UNICODE</td>
<td></td>
<td>Fixed-length UNICODE char data.</td>
</tr>
<tr>
<td>CT_NVARCHAR</td>
<td>CT_2UNICODE</td>
<td></td>
<td>Variable-length UNICODE char data.</td>
</tr>
</tbody>
</table>

*CT_BINARY field type was changed in c-treeACE V9 to CT_F2STRING.*

### c-treeDB .NET FIELD_TYPE Enum

All the c-treeDB .NET field types are defined using the FIELD_TYPE enum:

<table>
<thead>
<tr>
<th>c-treeDB .NET Field Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
</tr>
<tr>
<td>TINYINT</td>
</tr>
<tr>
<td>UTINYINT</td>
</tr>
<tr>
<td>SMALLINT</td>
</tr>
<tr>
<td>USMALLINT</td>
</tr>
<tr>
<td>INTEGER</td>
</tr>
<tr>
<td>UINTeger</td>
</tr>
<tr>
<td>MONEY</td>
</tr>
<tr>
<td>DATE</td>
</tr>
<tr>
<td>TIME</td>
</tr>
</tbody>
</table>
c-treeDB .NET Field Type

<table>
<thead>
<tr>
<th>Field Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOAT</td>
</tr>
<tr>
<td>DOUBLE</td>
</tr>
<tr>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>EFLOAT</td>
</tr>
<tr>
<td>BINARY</td>
</tr>
<tr>
<td>CHAR</td>
</tr>
<tr>
<td>FPSTRING</td>
</tr>
<tr>
<td>F2STRING</td>
</tr>
<tr>
<td>F4STRING</td>
</tr>
<tr>
<td>BIGINT</td>
</tr>
<tr>
<td>NUMBER</td>
</tr>
<tr>
<td>CURRNCY</td>
</tr>
<tr>
<td>PSTRING</td>
</tr>
<tr>
<td>VARBINARY</td>
</tr>
<tr>
<td>LVB</td>
</tr>
<tr>
<td>VARCHAR/LVC</td>
</tr>
</tbody>
</table>

```cpp
__value enum FIELD_TYPE {
    BOOL = CT_BOOL,
    CHAR = CT_CHAR,
    CHARU = CT_CHARU,
    INT2 = CT_INT2,
    INT2U = CT_INT2U,
    INT4 = CT_INT4,
    INT4U = CT_INT4U,
    MONEY = CT_MONEY,
    DATE = CT_DATE,
    TIME = CT_TIME,
    SFLOAT = CT_SFLOAT,
    DFLOAT = CT_DFLOAT,
    TIMES = CT_TIMES,
    EFLOAT = CT_EFLOAT,
    ARRAY = CT_ARRAY,
    FSTRING = CT_FSTRING,
    FPSTRING = CT_FPSTRING,
    F2STRING = CT_F2STRING,
    F4STRING = CT_F4STRING,
    BIGINT = CT_INT8,  //CT_BIGINT
    UBIGINT = CT_INT8U,  //CT_UBIGINT
    NUMBER = CT_NUMBER,
    CURRENCY = CT_CURRENCY,
    VSTRING = CT_STRING,
    VPSTRING = CT_PSTRING,
    V2STRING = CT_2STRING,
    V4STRING = CT_4STRING,
};
```
#ifdef ctUNICODE
    FUNICODE = CT_FUNICODE,
    CTUNICODE = CT_UNICODE,
    F2UNICODE = CT_F2UNICODE,
    V2UNICODE = CT_V2UNICODE,
#endif

TINYINT = CT_TINYINT,
UTINYINT = CT_UTINYINT,
SMALLINT = CT_SMALLINT,
USMALLINT = CT_USMALLINT,
INTEGER = CT_INTEGER,
UINTEGER = CT_UINTEGER,
FLOAT = CT_FLOAT,
DOUBL = CT_DOUBLE,
TIMESTAMP = CT_TIMESTAMP,
BINARY = CT_BINARY,
CHARS = CT_CHARS,
VARCHAR = CT_VARCHAR,
#endif ctUNICODE

NCHAR = CT_NCHAR,
NVARCHAR = CT_NVARCHAR,
#endif
LVC= CT_LVC,
VARBINARY = CT_VARBINARY,
LVB= CT_LVB
};

Find Modes

Use the following find modes with the record find methods:

<table>
<thead>
<tr>
<th>c-treeDB Find mode</th>
<th>c-treeDB .NET Find Mode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTFIND_EQ</td>
<td>EQ</td>
<td>Find a record equal to the target</td>
</tr>
<tr>
<td>CTFIND_LT</td>
<td>LT</td>
<td>Find a record less than target</td>
</tr>
<tr>
<td>CTFIND_LE</td>
<td>LE</td>
<td>Find a record less or equal than target</td>
</tr>
<tr>
<td>CTFIND_GT</td>
<td>GT</td>
<td>Find a record greater than target</td>
</tr>
<tr>
<td>CTFIND_GE</td>
<td>GE</td>
<td>Find a record greater or equal than target</td>
</tr>
</tbody>
</table>

**Note:** The Find Mode CTFIND_EQ requires that the target contains values for all segments that compose the index and the index cannot allow duplicates.

**Note:** c-treeDB .NET defines this mode with the FIND_MODE enum.
# Index Key Types

<table>
<thead>
<tr>
<th>c-treeDB Index Type</th>
<th>c-treeDB .NET Index Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTINDEX_FIXED</td>
<td>FIXED_INDEX</td>
<td>Fixed-length key</td>
</tr>
<tr>
<td>CTINDEX_LEADING</td>
<td>LEADING_INDEX</td>
<td>Fixed-length keys that are likely to have leading character duplication among the key values</td>
</tr>
<tr>
<td>CTINDEX_PADDING</td>
<td>PADDING_INDEX</td>
<td>Variable-length keys for which not much leading character duplication is expected.</td>
</tr>
<tr>
<td>CTINDEX_LEADPAD</td>
<td>LEADPAD_INDEX</td>
<td>Variable-length keys for which much leading character duplication is expected.</td>
</tr>
<tr>
<td>CTINDEX_ERROR</td>
<td>ERROR_INDEX</td>
<td>Index type error.</td>
</tr>
<tr>
<td>CTINDEX_DFRIDX</td>
<td>INDEX_DFRIDX</td>
<td>Indicates a deferred index (V11 and later).</td>
</tr>
<tr>
<td>CTINDEX_NOMOD</td>
<td>INDEX_NOMOD</td>
<td>Indicates an index with unmodifiable ISAM and c-treeDB keys (V11 and later).</td>
</tr>
</tbody>
</table>

**Note:** c-treeDB .NET Index Key Types are defined in the `KEY_TYPE` enum.

# Record Lock Modes

<table>
<thead>
<tr>
<th>c-treeDB Record Lock Modes</th>
<th>c-treeDB .NET Lock Modes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTLOCK_FREE</td>
<td>FREE_LOCK</td>
<td>Free the data record lock</td>
</tr>
<tr>
<td>CTLOCK_READ</td>
<td>READ_LOCK</td>
<td>Non-blocking read locks</td>
</tr>
<tr>
<td>CTLOCK_READ_BLOCK</td>
<td>READ_BLOCK_LOCK</td>
<td>Blocking read locks</td>
</tr>
<tr>
<td>CTLOCK_WRITE</td>
<td>WRITE_LOCK</td>
<td>Non-blocking write locks</td>
</tr>
<tr>
<td>CTLOCK_WRITE_BLOCK</td>
<td>WRITE_BLOCK_LOCK</td>
<td>Blocking write locks</td>
</tr>
</tbody>
</table>

**Note:** c-treeDB .NET can find the Lock Modes listed in the `LOCK_MODE` enum.

# Session Wide Lock Modes

<table>
<thead>
<tr>
<th>c-treeDB Lock Modes</th>
<th>c-treeDB .NET Lock Modes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTLOCK_FREE</td>
<td>FREE_LOCK</td>
<td>Free all locks. Free the data record lock.</td>
</tr>
<tr>
<td>CTLOCK_READ</td>
<td>READ_LOCK</td>
<td>Non-blocking read locks. If the lock cannot be acquired an error is returned.</td>
</tr>
<tr>
<td>CTLOCK_READ_BLOCK</td>
<td>READ_BLOCK_LOCK</td>
<td>Blocking read lock. The thread will block until the lock can be acquired.</td>
</tr>
</tbody>
</table>
c-treeDB Lock Modes | c-treeDB .NET Lock Modes | Explanation
--- | --- | ---
CTLOCK_WRITE | WRITE_LOCK | Non-blocking write lock. If the lock cannot be acquired an error is returned.
CTLOCK_WRITE_BLOCK | WRITE_BLOCK_LOCK | Blocking write lock. The thread will block until the lock can be acquired.
CTLOCK_RESET | RESET_LOCK | equivalent to calling Lock with CTLOCK_FREE followed by Lock() with CTLOCK_WRITE.
CTLOCK_SUSPEND | SUSPEND_LOCK | Temporarily suspend locking.
CTLOCK_RESTORE_READ | RESTORE_READ_LOCK | To be used after a call to Lock with the CTLOCK_SUSPEND mode. This lock mode restores the lock mode as READ.
CTLOCK_RESTORE_READ_BLOCK | RESTORE_READ_BLOCK_LOCK | To be used after a call to Lock with the CTLOCK_SUSPEND mode. This lock mode restores the lock mode as READ_BLOCK.
CTLOCK_RESTORE_WRITE | RESTORE_WRITE_LOCK | To be used after a call to Lock with the CTLOCK_SUSPEND mode. This lock mode restores the lock mode as WRITE.
CTLOCK_RESTORE_WRITE_BLOCK | RESTORE_WRITE_BLOCK_LOCK | To be used after a call to Lock with the CTLOCK_SUSPEND mode. This lock mode restores the lock mode as WRITE_BLOCK.
CTLOCK_RESTORE_PREVIOUS | | To be used after a call to Lock with the CTLOCK_SUSPEND mode. This lock mode restores the same lock mode valid before suspending the lock.

c-treeDB .NET Lock Modes are defined in the LOCK_MODE enum.

Segment Modes
The segment modes based on absolute field number, also known as schema fields, are the preferred modes to use in the segment definition. The preferred segment modes are:
- CTSEG_SCHSEG
- CTSEG_USCHSEG
- CTSEG_VSCHSEG
- CTSEG_UVSCHSEG
- CTSEG_SCHSRL

You may OR in the mode CTSEG_DESCENDING to the segment mode to specify the descending sort order for a segment. You can also or in the segment mode CTSEG_ALTSEG to specify an alternate collating sequence for the segment.

Using the preferred segment modes makes c-treeDB based tables fully compatible with ISAM/Low Level applications and/or c-treeACE SQL applications.
c-treeDB Segment Modes | c-treeDB .NET Segment Modes | Explanation
---|---|---
CTSEG_SCHSEG | SCHSEG_SEG | Absolute field number
CTSEG_USCHSEG | USCHSEG_SEG | Absolute field number - uppercase
CTSEG_VSCHSEG | VSCHSEG_SEG | Absolute field number - pad strings
CTSEG_UVSCHSEG | UVSCHSEG_SEG | Absolute field number - pad strings upper
CTSEG_SCHSRL | SCHSRL_SEG | Absolute field number - auto increment
CTSEG_DESCENDING | DESCENDING_SEG | Descending segment mode
CTSEG_ALTSEG | ALTSEG_SEG | Alternative collating sequence
CTSEG_ENDSEG | ENDSEG_SEG | END segment mode

The other segment modes are kept for compatibility with existing c-treeACE applications. Advanced c-treeDB functions like `ctdbAlterTable()` may not work properly if the segment mode is not one of the preferred segment modes.

You may specify these segment modes with `ctdbAddSegmentEx()`, which expects an absolute record offset where the segment is to start instead of a field indicator, the length in bytes of the segment, and the segment mode.

| c-treeDB Segment Modes | c-treeDB .NET Segment Modes | Explanation
---|---|---
CTSEG_REGSEG | REGSEG_SEG | Absolute byte offset - No transformation
CTSEG_INTSEG | INTSEG_SEG | Absolute byte offset - unsigned int/long
CTSEG_UREGSEG | UREGSEG_SEG | Absolute byte offset - uppercase
CTSEG_SRLSEG | SRLSEG_SEG | Absolute byte offset - auto increment
CTSEG_VARSEG | VARSEG_SEG | Relative field number
CTSEG_UVARSEG | UVARSEG_SEG | Relative field number - uppercase
CTSEG_SGNSEG | SGNSEG_SEG | Absolute byte offset - signed int/long
CTSEG_FLTSEG | FLTSEG_SEG | Absolute byte offset - float/double
CTSEG_DECSEG | DECSEG_SEG | Absolute byte offset - not yet implemented
CTSEG_BCDSEG | BCDSEG_SEG | Absolute byte offset - not yet implemented
CTSEG_DESCENDING | DESCENDING_SEG | Descending segment mode
CTSEG_ALTSEG | ALTSEG_SEG | Alternative collating sequence

c-treeDB .NET Segment Modes are defined in the `SET_MODE` enum.
### Table Create Modes

<table>
<thead>
<tr>
<th>c-treeDB Table Create Mode</th>
<th>c-treeDB .NET Table Create Mode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTCREATE_NORMAL</td>
<td>NORMAL_CREATE</td>
<td>Normal table creation. Use this mode when no other create mode apply.</td>
</tr>
<tr>
<td>CTCREATE_PREIMG</td>
<td>PREIMG_CREATE</td>
<td>This mode implements transaction processing for a table but does not support automatic file recovery. Files with CTCREATE_PREIMG mode do not take any space in the system transaction logs.</td>
</tr>
<tr>
<td>CTCREATE_TRNLOG</td>
<td>TRNLOG_CREATE</td>
<td>With this mode you will get the full benefit of transaction processing, including both atomicity and automatic recovery. If you are not sure of what mode to use, and you do want to use transaction processing, then use this mode.</td>
</tr>
<tr>
<td>CTCREATE_WRITETHRU</td>
<td>WRITETHRU_CREATE</td>
<td>This mode forces the operating system to flush all disk cache buffers when a data write occurs. Setting this mode can slow performance of the file handler. On the other hand, it is an important feature to use if you want to ensure that all data writes are put to the disk immediately. It is particularly important if you are in an operating environment where the system crashes a lot, and you are not using transactions. However, this mode does not guarantee that operating system buffers will be flushed as expected.</td>
</tr>
<tr>
<td>CTCREATE_CHECKLOCK</td>
<td>CHECKLOCK_CREATE</td>
<td>Tables created with this mode requires a record lock before a record can be updated. If a lock is not obtained, the error code DADV_ERR is returned.</td>
</tr>
<tr>
<td>CTCREATE_NORECBY</td>
<td>NORECBY_CREATE</td>
<td>Create the table without the RECBYT index.</td>
</tr>
<tr>
<td>CTCREATE_NOROWID</td>
<td>NOROWID_CREATE</td>
<td>Create the table without the ROWID index.</td>
</tr>
<tr>
<td>CTCREATE_CHECKREAD</td>
<td>CHECKREAD_CREATE</td>
<td>Tables create with this mode requires a record lock as records are read. Obtain at least a read lock on a record before it can be read, otherwise the function will return error code DADV_ERR.</td>
</tr>
<tr>
<td>CTCREATE_HUGEFILE</td>
<td>HUGEFILE_CREATE</td>
<td>Create the table with huge file support. With this mode on, tables will support 8 byte addresses for file offsets.</td>
</tr>
<tr>
<td>CTCREATE_NODELFLD</td>
<td>NODELFLD_CREATE</td>
<td>This mode indicate that the create is to be created without the $DELFLD$ field support.</td>
</tr>
<tr>
<td>CTCREATE_NONULFLD</td>
<td>NONULFLD_CREATE</td>
<td>This mode indicate that the table is to be created without the $NULFLD$ field support.</td>
</tr>
<tr>
<td>CTCREATE_COMPRESS</td>
<td>COMPRESS_CREATE</td>
<td>Creates tables with data compression support. When this mode is used, c-treeDB automatically creates the file as variable-length.</td>
</tr>
</tbody>
</table>
### Table Open Modes

<table>
<thead>
<tr>
<th>c-treeDB File Open Mode</th>
<th>c-treeDB .NET File Open Mode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTOPEN_NORMAL</td>
<td>NORMAL_OPEN</td>
<td>Use this mode if no other open modes apply.</td>
</tr>
</tbody>
</table>
| CTOPEN_DATAONLY        | DATAONLY_OPEN               | Open only the data table. Used to rebuild a table that may or may not be missing indices.  
  • **Caution**: Updates made to a data file with this file mode will not have any necessary updates reflected in the associated index files. |
<p>| CTOPEN_EXCLUSIVE       | EXCLUSIVE_OPEN              | This mode opens the table as exclusive. If this mode is used, only one user can open a table. If an application already has the file open in any mode, no other application can open the table as CTOPEN_EXCLUSIVE. Once an application opens a table as CTOPEN_EXCLUSIVE, no other application can open it. Reads and writes are cached for index files opened with this file mode since there are no integrity issues with only one process in the file. |
| CTOPEN_PERMANENT       | PERMANENT_OPEN              | Many operating systems and/or C compiler run-time libraries limit the number of files that can be opened at one time. A permanent file open causes the file to be opened and stay open until the program executes a file close. A non-permanent file open causes the table data and index files to be opened, but allows them to be transparently closed and reopened to allow other data and index files to be used. When it is necessary for a data and index file to be temporarily closed, c-treeACE selects the least recently used file. This file remains closed until it is used, at which time it will be automatically reopened. This strategy causes c-treeACE to use all available file descriptors. |</p>
<table>
<thead>
<tr>
<th>c-treeDB File Open Mode</th>
<th>c-treeDB .NET File Open Mode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTOPEN_CORRUPT</td>
<td>CORRUPT_OPEN</td>
<td>This mode opens tables with corrupted indices or in certain cases, tables with corrupted data. With c-treeDB this mode is usually used in conjunction with ctdbAlterTable() mode to perform a rebuild if the indices became corrupted: open table with CTOPEN_CORRUPT mode, then call ctdbAlterTable() with CTDB_ALTER_INDEX mode to force the rebuild of all indices of the table. You can also specify ctdbAlterTable() mode (CTDB_ALTER_INDEX CTDB_ALTER_PURGEDUP) to purge any duplicate records that may cause the index rebuild to fail. If a table table becomes corrupt, the table may be open with CTOPEN_CORRUPT mode and then ctdbAlterTable() with CTDB_ALL_FULL is invoked to try to recover the table.</td>
</tr>
<tr>
<td>CTOPEN_CHECKLOCK</td>
<td>CHECKLOCK_OPEN</td>
<td>Tables opened with this mode requires a record lock before a record can be updated. If a lock is not obtained, the error code DADV_ERR is returned.</td>
</tr>
<tr>
<td>CTOPEN_CHECKREAD</td>
<td>CHECKREAD_OPEN</td>
<td>Tables opened with this mode requires a record lock as records are read. Obtain at least a read lock on a record before it can be read, otherwise the function will return error code DADV_ERR.</td>
</tr>
<tr>
<td>CTOPEN_READONLY</td>
<td>READONLY_OPEN</td>
<td>Opens the table in READONLY mode and does not allow any modifications to the table structure or data records.</td>
</tr>
</tbody>
</table>

**Note:** c-treeDB .NET users can find the open modes listed in the OPEN_MODE enum.

### Table Permissions

<table>
<thead>
<tr>
<th>c-treeDB Permission Constant</th>
<th>c-treeDB .NET Permission Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPF_READ</td>
<td>O_READ</td>
<td>owner read permission</td>
</tr>
<tr>
<td>OPF_WRITE</td>
<td>O_WRITE</td>
<td>owner write/update permission</td>
</tr>
<tr>
<td>OPF_DEF</td>
<td>O_DEF</td>
<td>owner file definition permission</td>
</tr>
<tr>
<td>OPF_DELETE</td>
<td>O_DELETE</td>
<td>owner file deletion permission</td>
</tr>
<tr>
<td>OPF_ALL</td>
<td>O_ALL</td>
<td>owner granted all permissions</td>
</tr>
<tr>
<td>OPF_NOPASS</td>
<td>O_NOPASS</td>
<td>owner grants read only without password</td>
</tr>
<tr>
<td>GPF_NONE</td>
<td>G_NONE</td>
<td>group access denied</td>
</tr>
<tr>
<td>GPF_READ</td>
<td>G_READ</td>
<td>group read permission</td>
</tr>
<tr>
<td>c-treeDB Permission Constant</td>
<td>c-treeDB .NET Permission Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>GPF_WRITE</td>
<td>G_WRITE</td>
<td>group write/update permission</td>
</tr>
<tr>
<td>GPF_DEF</td>
<td>G_DEF</td>
<td>group file definition permission</td>
</tr>
<tr>
<td>GPF_DELETE</td>
<td>G_DELETE</td>
<td>group file deletion permission</td>
</tr>
<tr>
<td>GPF_NOPASS</td>
<td>G_NOPASS</td>
<td>group read only access without password</td>
</tr>
<tr>
<td>WPF_NONE</td>
<td>W_NONE</td>
<td>world access denied</td>
</tr>
<tr>
<td>WPF_READ</td>
<td>W_READ</td>
<td>world read permission</td>
</tr>
<tr>
<td>WPF_WRITE</td>
<td>W_WRITE</td>
<td>world write/update permission</td>
</tr>
<tr>
<td>WPF_DEF</td>
<td>W_DEF</td>
<td>world file definition permission</td>
</tr>
<tr>
<td>WPF_DELETE</td>
<td>W_DELETE</td>
<td>world file deletion permission</td>
</tr>
<tr>
<td>WPF_NOPASS</td>
<td>W_NOPASS</td>
<td>world read only access without password</td>
</tr>
</tbody>
</table>
4.2 CTBase Class

class CTBase

Description
The CTBase Class is the basic class, upon which most of the other classes are built.

See Also
CTSession, CTDatabase, CTTable

Preconditions
In general, this class is not directly used. However, since most of the other classes are based on this, there are some important methods described in here that are applicable to most of the derived classes.

CTBase Public Members

CTHANDLE m_handle
CTBase Methods

Constructor / Destructor
- `CTBase()`: Creates a `CTBase` object
- `~CTBase()`: Destroys a `CTBase` object and resets all the dependent objects

Error Handling
- `GetError()`: Returns the error code associated with the handle
- `SetError()`: Sets the error code
- `ClearError()`: Resets the error code

Transaction Processing
- `IsTransActive()`: Indicates if a transaction has been started with a call to `Begin()`, but not terminated with a call to `Commit` or `Abort`.
- `Begin()`: Begins a transaction.
- `Commit()`: Commits a transaction started with a call to `Begin()`.
- `Abort()`: Aborts a transaction started with a call to `Begin()`.
- `SetSavePoint()`: Sets a new transaction save point.
- `RestoreSavePoint()`: Restores a previously set save point.
- `Lock()`: Enables or disables record locks, using one of the c-tree Plus lock modes
- `Unlock()`: Disables record locks

Handle
- `GetHandle()`: Returns the handle
- `GetHandleId()`: Returns the handle ID
- `SetHandle()`: Sets the handle

Type Definition
- `GetDefFloatFormat()`: Retrieves the default floating point string format to be used in `sprintf()` and `sscanf()`
- `GetDefDateType()`: Retrieves the default date type
- `GetDefTimeType()`: Retrieves the default time type
- `SetDefFloatFormat()`: Sets the default floating point string format to be used in `sprintf()` and `sscanf()`
- `SetDefDateType()`: Sets the default date type
- `SetDefTimeType()`: Sets the default time type

Other
- `GetAutoCommit()`: Retrieves the c-treeDB auto commit mode.
- `GetUserTag()`: Retrieves the user tag.
- `SetUserTag()`: Sets the user tag.
- `SwitchInstance()`: Force a c-tree Plus instance switch.
- **GetKeepLock()**: Gets the Keep Lock mode in use.
- **SetKeepLock()**: Sets the Keep Lock mode in use.
CTBase::CTBase

Syntax
CTBase ( )

Parameters
This constructor has no parameters.

Description
This is the constructor for the CTBase class.

See also
~CTBase()
CTBase::~CTBase

Syntax
~CTBase

Parameters
This destructor has no parameters.

Description
This is the destructor for the CTBase class.

See also
CTBase()
CTBase::Abort

Syntax

void Abort ( )

Parameters

This method has no parameters.

Description

Aborts a transaction started with a call to Begin(). Note that Abort does not free any locks. Locks acquired during the transaction need to be released with a call to Unlock or CTRecord::UnlockRecord().

Return

None.

Example

pSession->Begin;
try {
    pDatabase->AddTable("custmast", "");
    printf("\nTable added to database.");
    pSession->Commit;
}
catch (CTException& err)
{
    Abort(  );
    printf("\n\n%d  %s", err.GetErrorCode, err.GetErrorMsg);
}

See also

Begin(), Commit(), IsTransActive(), SetKeepLock()
CTBase::GetAutoCommit

Declaration
CTBOOL CTBase::GetAutoCommit();

Description
CTBase::GetAutoCommit() retrieves the c-treeDB auto commit mode. The auto commit of transactions are invoked automatically when records are added or updated by CTRecord::Write() method.

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>YES</td>
<td>auto commit is enabled.</td>
</tr>
<tr>
<td>1</td>
<td>NO</td>
<td>auto commit is not enabled.</td>
</tr>
</tbody>
</table>

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

See Also
CTBase::SetAutoCommit() CTBase::SetOperation() CTBase::GetOperation()
CTBase::Begin

Syntax
void Begin ( )

Parameters
This method has no parameters.

Description
Marks the beginning of a transaction. If all the operations intended for the particular transaction are finished correctly, use Commit() to commit the transaction. Otherwise, use Abort() to abort the transaction.

The Begin() method does not lock or set the flags to lock any records. In order to request the lock of the records to be updated inside the transaction, the Lock() or CTRrecord::LockRecord() should be used.

Return
None.

Example
pSession->Begin;
try {
    pDatabase->AddTable("custmast", "");
    printf("\nTable added to database.");
    pSession->Commit;
}
catch (CTException& err)
{
    Abort( );
    printf("\n\n%d  %s", err.GetErrorCode, err.GetErrorMsg);
}

See also
Abort(), Commit(), IsTransActive(), Lock()
CTBase::ClearError

Syntax
void ClearError ( )

Parameters
This method has no parameters.

Description
Clear the current error and set the error to CTDBRET_OK.

Return
None.

See also
getError(), setError()
CTBase::Commit

Syntax

void Commit ( )

Parameters

This method has no parameters.

Description

Commits a transaction started with a call to Begin(). Notice that Commit does not free any locks. Locks acquired during the transaction need to be released with a call to Unlock() or CTRecord::UnlockRecord().

Return

None.

Example

pSession->Begin;
try {
    pDatabase->AddTable("custmast", "");
    printf("nTable added to database.");
    pSession->Commit;
}
catch (CTException& err)
{
    Abort(  );
    printf("\n\n%d  %s", err.GetErrorCode, err.GetErrorMsg);
}

See also

Begin(), Abort(), IsTransActive(), SetKeepLock()
CTBase::GetDefDateType

Syntax
CTDATE_TYPE GetDefDateType() const

Parameters
This method has no parameters.

Description
Retrieves the default date type. The valid values for the date type are shown in "Data Types" (page 163).

Return
GetDefDateType() returns the default date type. For more information on the valid values for the date type, see the description above.

See also
SetDefDateType()
CTBase::GetDefFloatFormat

Syntax

CTString GetDefFloatFormat()

Parameters

This method has no parameters.

Description

Retrieves the floating point string format to be used as a default in `sprintf()` and `sscanf()`. The default value initially is set to "%f", and may be changed to reflect the desired printing format.

Return

`GetDefFloatFormat()` returns the default floating point format.

See also

`SetDefFloatFormat()`
CTBase::GetDefTimeType

Syntax

CTTIME_TYPE GetDefTimeType ( ) const

Parameters

This method has no parameters.

Description

Retrieves the default time type. The valid values for the time type are shown in "Data Types" (page 163).

Return

GetDefTimeType() returns the default time type. For more information on the valid values for the time type, see the description above.

See also

SetDefTimeType()
CTBase::GetError

Syntax
CTDBRET GetError ( )

Parameters
This method has no parameters.

Description
Returns the error code associated with the handle.

Return
GetError() returns the error code associated with the handle or CTDBRET_NULHANDLE (4002)
if the handle is unallocated.

See also
setError(), ClearError()
CTBase::GetHandle

Syntax
CTHANDLE GetHandle ( ) const

Parameters
This method has no parameters.

Description
Returns the handle.

Return
GetHandle() returns the c-treeDB C API handle

See also
GetHandleId(), SetHandle()
CTBase::GetHandleId

Syntax
NINT GetHandleId() const

Parameters
This method has no parameters.

Description
Returns the handle ID.

Return
GetHandleId() returns the handle ID or 0 on error.

See also
GetHandle(), SetHandle()
CTBase::GetKeepLock

Syntax
CTKEEP_MODE CTBase::GetKeepLock();

Parameters
This method has no parameters.

Description
When a transaction is terminated by calling either the CTBase::Commit() or CTBase::Abort() function, all locks are automatically freed, including the locks acquired outside the transaction. An application might want to control which locks are kept when the transaction ends. To support this ability, c-treeDB introduced a function to control how locks are handled when a transaction terminates.

CTBase::GetKeepLock() returns the current keep lock status.

Return

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CTKEEP_FREE</td>
<td>Release all locks. Clear LKISAM state. This is the default mode.</td>
</tr>
<tr>
<td>1</td>
<td>CTKEEP_LOCK</td>
<td>Keep all locks acquired before and during transaction. The LKISAM state is not cleared.</td>
</tr>
<tr>
<td>2</td>
<td>CTKEEP_OUT</td>
<td>Release only locks obtained within transaction and/or locks on records updated within transaction. The LKISAM state is not cleared.</td>
</tr>
<tr>
<td>3</td>
<td>CTKEEP_OUTALL</td>
<td>Unconditionally keep all locks acquired before transaction began. Free locks obtained within the transaction. The LKISAM state is not cleared.</td>
</tr>
</tbody>
</table>

See also
CTBase::Abort(), CTBase::Begin(), CTBase::Commit(), CTBase::SetKeepLock()
CTBase::GetLockMode

Syntax
CTLOCK_MODE GetLockMode ( )

Parameters
This method has no parameters.

Description
Retrieves the current record lock mode for session wide locks. Valid values for the session wide lock mode are shown in "Session Wide Lock Modes" (page 231). If CTLOCK_FREE is returned, it does indicate that no locks are currently active.

Return
GetLockMode() returns the session wide lock mode, as described in "Session Wide Lock Modes" (page 231).

See also
Lock(), Unlock(), IsLockActive()
CTBase::GetOperation

Syntax
CTOPS_MODE CTBase::GetOperation();

Description
GetOperation() retrieves the operation modes for special performance-related functionality and test operational states for critical events.

Return
GetOperation() return the current operations state, which is a combination of the following modes:

<table>
<thead>
<tr>
<th>Operations Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPS_READLOCK</td>
<td>Enable automatic, low level, blocking read locks on each record access that does not already have a lock.</td>
</tr>
<tr>
<td>OPS_LOCKON_GET</td>
<td>Lock next fetch only.</td>
</tr>
<tr>
<td>OPS_UNLOCK_ADD</td>
<td>Automatic unlock on add.</td>
</tr>
<tr>
<td>OPS_UNLOCK_RWT</td>
<td>Automatic unlock on rewrite.</td>
</tr>
<tr>
<td>OPS_UNLOCK_UPD</td>
<td>(OPS_UNLOCK_ADD</td>
</tr>
<tr>
<td>OPS_LOCKON_BLK</td>
<td>Blocking lock on next fetch only.</td>
</tr>
<tr>
<td>OPS_FUNCTION_MON</td>
<td>Toggle function monitor. (Server)</td>
</tr>
<tr>
<td>OPS_LOCK_MON</td>
<td>Toggle lock monitor. (Server)</td>
</tr>
<tr>
<td>OPS_TRACK_MON</td>
<td>Toggle memory track monitor. (Server)</td>
</tr>
<tr>
<td>OPS_MIRROR_NOSWITCH</td>
<td>Don’t continue if mirror or primary fails. (Server)</td>
</tr>
<tr>
<td>OPS_MIRROR_TRM</td>
<td>A primary or mirror has been shutdown.</td>
</tr>
<tr>
<td>OPS_MEMORY_SWP</td>
<td>Memory swapping active.</td>
</tr>
<tr>
<td>OPS_AUTOISAM_TRN</td>
<td>Automatic ISAM transactions.</td>
</tr>
<tr>
<td>OPS_KEEPLOK_TRN</td>
<td>Keep locks involved in automatic transactions on record adds and updates after commit.</td>
</tr>
<tr>
<td>OPS_SERIAL_UPD</td>
<td>Changes GetSerialNbr() operation.</td>
</tr>
<tr>
<td>OPS_DEFER_CLOSE</td>
<td>Defer file closes or deletes during transactions.</td>
</tr>
<tr>
<td>OPS_CONV_STRING</td>
<td>Change all CT_STRING fields having a non-zero field length in the fixed length portion of the record buffer to CT_FSTRING fields. (Client)</td>
</tr>
<tr>
<td>OPS_DISK_IO</td>
<td>Set sysiocod on disk reads and writes.</td>
</tr>
</tbody>
</table>

See Also
CTBase::SetOperation CTBase::GetAutoCommit CTBase::SetAutoCommit
CTBase::GetSystemConfig

Retrieve c-tree Plus system configuration values.

Syntax
LONG CTBase::GetSystemConfig(NINT index);

Description
GetSystemConfig() retrieves c-tree Plus system configuration values, as well as some of the important dynamic aspects of the system, such as the memory usage and the number of files in use. To determine if a particular system configuration option is active, call GetSystemConfig(), passing the corresponding pre-define constant for that option, and check if the value returned is non-zero.

The following pre-defined constant should be passed to GetSystemConfig():

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfgMEMORY_USAGE</td>
<td>Current system memory usage.</td>
</tr>
<tr>
<td>cfgMEMORY_HIGH</td>
<td>Highest system memory use.</td>
</tr>
<tr>
<td>cfgNET_ALLOCS</td>
<td>Current system net allocations.</td>
</tr>
<tr>
<td>cfgOPEN_FILES</td>
<td>c-treeACE files opened by system.</td>
</tr>
<tr>
<td>cfgPHYSICAL_FILES</td>
<td>Physical c-treeACE files open. Includes c-tree Superfile members omitted from cfgOPEN_FILES count.</td>
</tr>
<tr>
<td>cfgOPEN_FCBS</td>
<td>c-treeACE file control blocks in use by system.</td>
</tr>
<tr>
<td>cfgLOGIDX</td>
<td>Is file mode ctLOGIDX supported?</td>
</tr>
</tbody>
</table>

The following constants only apply to client-server implementations:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfgDNODE_QLENGTH</td>
<td>Messages in delete node queue.</td>
</tr>
<tr>
<td>cfgCHKPNT_QLENGTH</td>
<td>Messages in checkpoint queue.</td>
</tr>
<tr>
<td>cfgSYSMON_QLENGTH</td>
<td>Messages in system monitor queue.</td>
</tr>
<tr>
<td>cfgLOGONS</td>
<td>Current number of logons.</td>
</tr>
<tr>
<td>cfgNET_LOCKS</td>
<td>Current number of pending locks (system wide).</td>
</tr>
<tr>
<td>cfgUSERS</td>
<td>Maximum number of logons.</td>
</tr>
<tr>
<td>cfgMAX_CONNECT</td>
<td>The limit for the maximum number of logons.</td>
</tr>
<tr>
<td>cfgUSER_FILES</td>
<td>Number of c-treeACE files opened by calling user.</td>
</tr>
<tr>
<td>cfgUSER_MEMORY</td>
<td>Current user memory usage.</td>
</tr>
<tr>
<td>cfgPATH_SEPARATOR</td>
<td>ASCII value for the file name path separator.</td>
</tr>
</tbody>
</table>

The following constants are static compile time values:
### Constant API Class Reference

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cfgFILES</code></td>
<td>Maximum number of c-tree Plus file control blocks available system wide.</td>
</tr>
<tr>
<td><code>cfgMAX_DAT_KEY</code></td>
<td>Maximum number of indices per data file.</td>
</tr>
<tr>
<td><code>cfgMAX_KEY_SEG</code></td>
<td>Maximum number of key segments per index.</td>
</tr>
</tbody>
</table>

The constants above and the pre-initialization resources section below have client and c-tree Server versions, except for the following three subscripts:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cfgINIT_CTREEapp</code></td>
<td>Determine whether c-treeACE has been initialized.</td>
</tr>
<tr>
<td><code>cfgSERIALNBR</code></td>
<td>The c-tree Server serial number.</td>
</tr>
<tr>
<td><code>cfgTHREADapp</code></td>
<td>Indicates if threading has been enabled.</td>
</tr>
</tbody>
</table>

Constants ending with ‘app’ are specific to the client side of a client/server application. To check the same system setting for the c-tree Server, use the same subscript without the app extension. For example, to determine if Conditional Index support is active on the c-tree Server, use `cfgCONIDX` as the subscript and `cfgCONIDXapp` for the client side.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cfgBOUNDapp</code></td>
<td>Indicates if the application is bound to a database library. See the discussion on the different FairCom I/O models in these notes.</td>
</tr>
<tr>
<td><code>cfgDISKIO_MODELapp</code></td>
<td>A non-zero value indicates a stand-alone multi-user I/O model i.e. FPUTFGET.</td>
</tr>
<tr>
<td><code>cfgLOCLIBapp</code></td>
<td>A non-zero value indicates Local Library support.</td>
</tr>
<tr>
<td><code>cfgNOGLOBALSapp</code></td>
<td>A non-zero value indicates no globals are supported, that is, indicating all globals are stored in an allocated structure. This is the default setting.</td>
</tr>
<tr>
<td><code>cfgUNIFRMAp</code></td>
<td>A non-zero value indicates FairCom's automatic byte flipping (UNIFRMAp) is active.</td>
</tr>
</tbody>
</table>

The pre-initialization resource constants below may be specified prior to a c-treeDB initialization call, i.e `ctdbLogon()`, `CTSession::Logon()`, or `CTSession.Logon()`, in addition to having both a client and c-tree Server version, as discussed above.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cfgANSIapp</code></td>
<td>Specifies whether to use ANSI. A non-zero value indicates ANSI.</td>
</tr>
<tr>
<td><code>cfgCONIDXapp</code></td>
<td>A non-zero value indicates the application supports. FairCom's Conditional Index Logic.</td>
</tr>
<tr>
<td><code>cfgCTBATCHapp</code></td>
<td>A non-zero value indicates the application supports. Batch Operations.</td>
</tr>
<tr>
<td><code>cfgCTSUPERapp</code></td>
<td>A non-zero value indicates the application supports c-tree Superfiles.</td>
</tr>
<tr>
<td><code>cfgCTS_ISAMapp</code></td>
<td>A non-zero value indicates the application supports FairCom's ISAM</td>
</tr>
<tr>
<td>Constant</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cfgHISTORYapp</td>
<td>A non-zero value indicates the application supports FairCom’s History Logic.</td>
</tr>
<tr>
<td>cfgINIT_CTREEapp</td>
<td>A non-zero value indicates c-treeACE has been initialized.</td>
</tr>
<tr>
<td>cfgLOGIDXapp</td>
<td>A non-zero value indicates the application supports the ctLOGIDX Logic.</td>
</tr>
<tr>
<td>cfgPARMFILEapp</td>
<td>A non-zero value indicates parameter files are supported.</td>
</tr>
<tr>
<td>cfgPASCAL24app1</td>
<td>A non-zero value indicates 2-byte/4-byte length delimited strings are using the traditional pascal length convention.</td>
</tr>
<tr>
<td>cfgPASCALstapp1</td>
<td>A non-zero value indicates byte length delimited strings are using the traditional pascal length convention.</td>
</tr>
<tr>
<td>cfgPATH_SEPARATORapp</td>
<td>Return the ASCII value for the file name path separator.</td>
</tr>
<tr>
<td>cfgPROTOTYPEapp</td>
<td>A non-zero value indicates the application supports Prototypes.</td>
</tr>
<tr>
<td>cfgRESOURCEapp</td>
<td>A non-zero value indicates the application supports Resource Records.</td>
</tr>
<tr>
<td>cfgRTREEapp</td>
<td>A non-zero value indicates the application supports the r-tree report engine.</td>
</tr>
<tr>
<td>cfgSERIALNBR</td>
<td>Return c-tree Server serial number.</td>
</tr>
<tr>
<td>cfgTHREADapp</td>
<td>A non-zero value indicates the application supports FairCom’s threading API.</td>
</tr>
<tr>
<td>cfgTRANPROCapp</td>
<td>A non-zero value indicates the application supports Transaction Processing.</td>
</tr>
<tr>
<td>cfgVARLDATAapp</td>
<td>A non-zero value indicates the application supports Variable Length Records.</td>
</tr>
<tr>
<td>cfgVARLKEYSapp</td>
<td>A non-zero value indicates the application supports Variable Length Keys i.e. Key compression.</td>
</tr>
<tr>
<td>cfgWORD_ORDERapp</td>
<td>Indicates the client data order: Low_High or High_Low. A non-zero value indicates Low_High.</td>
</tr>
</tbody>
</table>

1Pascal length byte
c-treeACE supports two different methods for specifying the length byte in a pascal data type. The original and non-traditional approach does not include the length byte in the byte count. For example, with a 1-byte data type, `CT_FPSTRING`, the smallest valid length byte would be 0. The new method follows the more traditional pascal convention of including the length byte in the byte count. For example, with the traditional approach, the smallest valid length for a 1-byte data type would be 1. Therefore, if `cfgPASCALstapp` or `cfgPASCAL24app` return a non-zero value, the new traditional approach is active.

To receive a valid return value from `cfgPATH_SEPARATOR`, `ctPATH_SEP` must be defined. The default definition found in `ctopt2.h` is:

```c
#define ctPATH_SEP   '?'
```
This definition specifies that the system default path separator will be used. To use the same separator for all platforms, you might want to choose one of the following:

```c
#define ctPATH_SEP  '\\'
/* define for Windows */
```
```
#define ctPATH_SEP  '/'
/* define for most Unix systems */
```
```
#define ctPATH_SEP  ':
/* define for Mac OSX */
```

/* where the Uninitialized value is NINT_ERR for a client or FINT_ERR for a bound application */

The following constants can be used to capture system-wide cache and buffer statistics, (cache pages hold data record images and buffers hold index nodes), allowing an application to track the use of these resources.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfgCACHE_PAGES</td>
<td>Available cache pages</td>
</tr>
<tr>
<td>cfgCACHE_PAGES_INUSE</td>
<td>Cache pages in use</td>
</tr>
<tr>
<td>cfgCACHE_PAGES_MXUSE</td>
<td>Maximum cache pages used</td>
</tr>
<tr>
<td>cfgCACHE_PAGES_DED</td>
<td>Available dedicated cache pages</td>
</tr>
<tr>
<td>cfgCACHE_PAGES_DEDINUSE</td>
<td>Dedicated cache pages in use</td>
</tr>
<tr>
<td>cfgCACHE_PAGES_DEDMXUSE</td>
<td>Maximum dedicated cache pages used</td>
</tr>
<tr>
<td>cfgBUFFER_PAGES</td>
<td>Available index buffers</td>
</tr>
<tr>
<td>cfgBUFFER_PAGES_INUSE</td>
<td>Index buffers in use</td>
</tr>
<tr>
<td>cfgBUFFER_PAGES_MXUSE</td>
<td>Maximum index buffers used</td>
</tr>
</tbody>
</table>

**Return**

Returns a value that depends on the configuration constant passed to `GetSystemConfig()`.

**Example**

```c
// check if TRANPROC was turned
if (hSession.GetSystemConfig(ctTRANPROC))
    printf("TRANPROC was turned on during compilation\n");
else
    printf("TRANPROC was turned off during compilation\n");
```

**See Also**

none
CTBase::GetTransactionMode

Declaration
CTBEGIN_MODE CTBase::GetTransactionMode()

Description
Returns the transaction mode used when starting a transaction.

c-tree Plus offers a rich array of data integrity options. Full transaction processing offers the safest and best performance of all the available options. There are times when other c-tree Plus options, such as PREIMG, might be advantageous.

Return Values

<table>
<thead>
<tr>
<th>CTBEGIN_MODE Symbolic Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTBEGIN_NONE</td>
<td>No begin transaction mode set. Default mode apply.</td>
</tr>
<tr>
<td>CTBEGIN_PREIMG</td>
<td>Transaction atomicity only. Auto-recovery is not available. Mutually exclusive with CTBEGIN_TRNLOG.</td>
</tr>
<tr>
<td>CTBEGIN_TRNLOG</td>
<td>Full transaction processing functionality including auto-recovery. Mutually exclusive to CTBEGIN_PREIMG. This is the default begin transaction mode.</td>
</tr>
<tr>
<td>CTBEGIN_DEFER</td>
<td>Defer begin transaction until update.</td>
</tr>
<tr>
<td>CTBEGIN_AUTOSAVE</td>
<td>Automatically invokes savepoints after each successful record or resource update.</td>
</tr>
</tbody>
</table>

See Also
CTBase::GetTransactionMode()
CTBase::GetUserTag

**Syntax**

pVOID GetUserTag ( )

**Parameters**

This method has no parameters.

**Description**

Retrieves the user tag.

**Return**

GetUserTag() returns the user tag.

**See also**

SetUserTag()
CTBase::IsLockActive

Syntax

CTBOOL IsLockActive ( )

Parameters

This method has no parameters.

Description

Indicates whether a session wide lock is active. A session wide lock is a lock that has been started with a call to Lock, with one of the READ, WRITE, and RESTORE mode variations, but not terminated with a call to Unlock or Lock with the mode parameter as FREE or SUSPEND.

IsLockActive() does not indicate if record lock is active. Record lock is one that is obtained with a call to CTRecord::LockRecord().

Return

IsLockActive() returns YES if the session wide lock is active and NO otherwise.

See also

Lock(), Unlock(), GetLockMode(), IsTransActive()
CTBase::IsTransActive

Syntax

CTBOOL IsTransActive ( ) const

Parameters

This method has no parameters.

Description

Indicates if a transaction has been started with a call to Begin, but not terminated with a call to Commit() or Abort().

Return

IsTransActive() returns YES if there is one transaction pending and NO otherwise.

See also

Begin(), Abort(), Commit(), IsLockActive()
CTBase::Lock

Syntax

```cpp
void Lock(CTLOCK_MODE mode)
```

Parameters

- `mode` [in] the lock mode. The valid values for mode are shown in "c-treeDB definitions" (page 227).

Description

Enables or disables session wide locks.

After a call to Lock with any mode but `CTLOCK_FREE`, any record that becomes a current record will be locked. For instance, all records read after a call to Lock will be locked. Notice that in the moment Lock is called, no file or record is locked. Instead, a flag is set internally to indicate that all new records read will lock the record. To simply lock the current record, use `CTRecord::LockRecord()`.

A record that is locked with one of the READ locks available allows any other user with a READ lock to read that record. No one can update a record using a READ lock. When a user gets a WRITE lock in a record, it means he/she may update that particular record, and until the record is freed, no one else is able to obtain any lock on that particular record. Yet anyone can still read the record.

If a record is read without any lock, it means that any other user can change the record contents. If a record is read with a READ lock, it means that any other user may still read the same record, but no user may obtain a WRITE lock, assuring the record will not be changed while a READ lock is kept on the record. When a record is read with a WRITE lock, the user who obtained the lock may update the record, but no other user may obtain a lock on the record.

A lock obtained with a call to `CTBase::Lock()`, no matter what object does the call, is always a Session lock, in the sense that any table operation after this call is subject to the locks.

A call to `Abort()`, `Commit()`, or `Unlock()` will release all session wide locks. A call to Lock with `CTLOCK_FREE` also releases the session wide locks.

Return

None.

Example

```cpp
pRecord1->Lock();
pRecord1->SetFieldAsString("name", "Johny Wild");
pRecord1->Find(CTFIND_EQ);
```

See also

- `Unlock()`, `IsLockActive()`, `GetLockMode()`, `CTRecord::LockRecord()`, `CTBase::SetKeepLock()`
CTBase::RestoreSavePoint

Syntax
void RestoreSavePoint (const NINT SavePoint)

Parameters
- SavePoint [in] a previously saved savepoint.

Description
Restore a previously set save point.

Return
None.

See also
SetSavePoint()
CTBase::SetAutoCommit

Declaration

void CTBase::SetAutoCommit(CTBOOL flag);

Description

CTBase::SetAutoCommit() sets the c-treeDB auto commit mode.

- flag indicates if auto commit mode should be enabled or not.

The auto commit of transactions are invoked automatically when records are added or updated by a `CTRecord::Write()` call. Automatic transactions may increase performance by reducing network traffic for single record updates since this is equivalent of performing the following functions:

```c
hRecord.Begin();
try {
    hRecord.Write();
    hRecord.Commit();
} catch (CTException) {
    hRecord.Abort();
}
```

The following applies when an automatic transaction is performed:

1. An automatic transaction will only free locks acquired by the `CTRecord::Write()` function.
2. If an automatic transaction is aborted because of errors detected by `CTRecord::Write()` all locks acquired by `CTRecord::Write()` are released.
3. If locks are active, by calling `CTBase::Lock()` function, an automatic transaction will not release any locks when it commits or aborts the `CTRecord::Write()` operation.
4. If OPS_UNLOCK_UPD is on, both commits and aborts on automatic transactions release only locks obtained within trans and/or locks on records updated within transaction, regardless if the c-treeDB session lock state is active or not.

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CTDBRET_OK</td>
<td>No Error</td>
</tr>
</tbody>
</table>

Example

```c
/*
   Example Code
*/
```

See Also

CTBase::GetAutoCommit() CTBase::SetOperation() CTBase::GetOperation()
CTBase::SetDefDateType

Syntax
void SetDefDateType (CTDATE_TYPE DateType)

Parameters
- *DataType* [in] the new default date type. For more information on the valid date types, see the Description below.

Description
Sets the default date type. The valid values for the date type are discussed in "Data Types" (page 163).

Return
None.

See also
GetDefDateType()
CTBase::SetDefFloatFormat

Syntax
void SetDefFloatFormat(const CTString& format)

Parameters
- format [in] the default floating point string format.

Description
Sets the default floating point string format to be used in sprintf(). The default value initially is set to "%f", and may be changed to reflect the desired printing format.

Return
None.

See also
GetDefFloatFormat()
CTBase::SetDefTimeType

Syntax
void SetDefTimeType(CTTIME_TYPE TimeType)

Parameters
- **TimeType** [in] The new default time type. For more information on the valid time types, see the Description below.

Description
Sets the default time type. The valid values for the time type are shown in "Data Types" (page 163).

Return
None.

See also
GetDefTimeType()
CTBase::SetError

Syntax

void SetError(const CTDBRET code)

Parameters

- code [in] the new error code.

Description

Sets the error code.

Return

None.

See also

GetError(), ClearError()
CTBase::SetHandle

Syntax
void SetHandle(CTHANDLE Handle)

Parameters
- Handle [in] is the object. Depending on the instance, it can be the Session, Database, Table, Record, Field, Index or Segment object.

Description
Sets the object.

Return
None.

See also
GetHandleId(), GetHandleId()
CTBase::SetKeepLock

Set the current keep lock mode when a transaction is committed or aborted.

Syntax

```cpp
void CTBase::SetKeepLock(CTKEEP_MODE mode);
```

Description

When a transaction is terminated by calling either the `CTBase::Commit()` or `CTBase::Abort()` function, all locks are automatically freed, including the locks acquired outside the transaction. An application might want to control which locks are kept when the transaction ends. To support this ability, c-treeDB introduced a function to control how locks are handled when a transaction terminates.

By default, the c-treeDB begin transaction function, `ctdbBegin()`, will begin a transaction by invoking c-tree’s `TRANBEG` function with the `ctTRNLOG` and `ctENABLE_BLK` modes set. If the `TRANBEG` function succeeds, c-treeDB’s `ctdbBegin()` function automatically calls `LKISAM()` to suspend locks enabled by `TRANBEG`, allowing users to enable any locks they wish: read locks, write locks, blocking or non-blocking. In this case, the c-treeDB commit or abort transaction functions, `ctdbCommit()` and `ctdbAbort()`, call the appropriate c-tree ISAM functions to terminate the transaction and free all locks.

`CTBase::SetKeepLock()` set the extended keep lock mode applied when an active transaction is committed or aborted.

The keep lock mode is one of the following pre-defined constants:

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CTKEEP_FREE</td>
<td>Release all locks. Clear LKISAM state. This is the default mode.</td>
</tr>
<tr>
<td>1</td>
<td>CTKEEP_LOCK</td>
<td>Keep all locks acquired before and during transaction. The LKISAM state is not cleared.</td>
</tr>
<tr>
<td>2</td>
<td>CTKEEP_OUT</td>
<td>Release only locks obtained within transaction and/or locks on records updated within transaction. The LKISAM state is not cleared.</td>
</tr>
<tr>
<td>3</td>
<td>CTKEEP_OUTALL</td>
<td>Unconditionally keep all locks acquired before transaction began. Free locks obtained within the transaction. The LKISAM state is not cleared.</td>
</tr>
</tbody>
</table>

Once the keep lock mode is set, it remains active until another keep mode is set or the keep mode is cleared by calling `SetKeepLock()` with mode `CTKEEP_FREE`.

Return

`void`

Example

```cpp
// set the keep lock
hSession.SetKeepLock(CTKEEP_LOCK);
```
// begin a new transaction
hSession.Begin();

// do some operations and commit transaction
hSession.Commit();

// do some more operations and free all locks
hSession.Unlock();

See Also
CTBase::Abort(), CTBase::Begin(), CTBase::Commit(), CTBase::GetKeepLock()
**CTBase::SetOperation**

**Syntax**

```cpp
void CTBase::SetOperation(CTOPS_MODE mode, CTOPS_STATE state);
```

**Description**

`SetOperation()` sets c-tree operation modes for special performance-related functionality and test operational states for critical events. 

*mode* use a combination of the following values:

<table>
<thead>
<tr>
<th>Operations Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OPS_READLOCK</code></td>
<td>Enable automatic, low level, blocking read locks on each record access that does not already have a lock.</td>
</tr>
<tr>
<td><code>OPS_LOCKON_GET</code></td>
<td>Lock next fetch only.</td>
</tr>
<tr>
<td><code>OPS_UNLOCK_ADD</code></td>
<td>Automatic unlock on add.</td>
</tr>
<tr>
<td><code>OPS_UNLOCK_RWT</code></td>
<td>Automatic unlock on rewrite.</td>
</tr>
<tr>
<td><code>OPS_UNLOCK_UPD</code></td>
<td>`OPS_UNLOCK_ADD</td>
</tr>
<tr>
<td><code>OPS_LOCKON_BLK</code></td>
<td>Blocking lock on next fetch only.</td>
</tr>
<tr>
<td><code>OPS_FUNCTION_MON</code></td>
<td>Toggle function monitor. (Server)</td>
</tr>
<tr>
<td><code>OPS_LOCK_MON</code></td>
<td>Toggle lock monitor. (Server)</td>
</tr>
<tr>
<td><code>OPS_TRACK_MON</code></td>
<td>Toggle memory track monitor. (Server)</td>
</tr>
<tr>
<td><code>OPS_MIRROR_NOSWITCH</code></td>
<td>Don’t continue if mirror or primary fails. (Server)</td>
</tr>
<tr>
<td><code>OPS_MIRROR_TRM</code></td>
<td>A primary or mirror has been shutdown.</td>
</tr>
<tr>
<td><code>OPS_MEMORY_SWP</code></td>
<td>Memory swapping active.</td>
</tr>
<tr>
<td><code>OPS_AUTOISAM_TRN</code></td>
<td>Automatic ISAM transactions.</td>
</tr>
<tr>
<td><code>OPS_KEEPLOK_TRN</code></td>
<td>Keep locks involved in automatic transactions on record adds and updates after commit.</td>
</tr>
<tr>
<td><code>OPS_SERIAL_UPD</code></td>
<td>Changes <code>GetSerialNbr()</code> operation.</td>
</tr>
<tr>
<td><code>OPS_DEFER_CLOSE</code></td>
<td>Defer file closes or deletes during transactions.</td>
</tr>
<tr>
<td><code>OPS_CONV_STRING</code></td>
<td>Change all <code>CT_STRING</code> fields having a non-zero field length in the fixed length portion of the record buffer to <code>CT_FSTRING</code> fields. (Client)</td>
</tr>
<tr>
<td><code>OPS_DISK_IO</code></td>
<td>Set <code>sysioked</code> on disk reads and writes.</td>
</tr>
</tbody>
</table>

*state* must be set with one of the following state values:

<table>
<thead>
<tr>
<th>Operation State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OPS_STATE_OFF</code></td>
<td>Turn a status bit off.</td>
</tr>
<tr>
<td><code>OPS_STATE_SET</code></td>
<td>Set the entire status word.</td>
</tr>
<tr>
<td><code>OPS_STATE_ON</code></td>
<td>Turn a status bit on.</td>
</tr>
<tr>
<td><code>OPS_STATE_RET</code></td>
<td>Return the entire status word.</td>
</tr>
</tbody>
</table>
### Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO_ERROR</td>
<td>No error occurred.</td>
</tr>
</tbody>
</table>

**See Also**

`CTBase::GetOperation()`
CTBase::SetSavePoint

Syntax
NINT SetSavePoint ( )

Parameters
This method has no parameters.

Description
Sets a new transaction save point, and returns this value.

Return
SetSavePoint() returns an integer with the save point number.

See also
RestoreSavePoint()
CTBase::SetUserTag

Syntax
void SetUserTag (pVOID tagptr)

Parameters
- *tagptr* [in] is the new tag to be set.

Description
Sets the user tag.

Return
None.

See also
GetUserTag()
CTBase::SetTransactionMode

Declaration
void CTBase::SetTransactionMode(CTBEGIN_MODE mode)

Description
c-tree Plus offers a rich array of data integrity options. Full transaction processing offers the safest and best performance of all the available options. There are times when other c-tree Plus options, such as PREIMG, might be advantageous.

Sets the transaction mode used when starting a transaction.

- **mode** is one of the following values:

<table>
<thead>
<tr>
<th>CTBEGIN_MODE Symbolic Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTBEGIN_NONE</td>
<td>No begin transaction mode set. Default mode apply.</td>
</tr>
<tr>
<td>CTBEGIN_PREIMG</td>
<td>Transaction atomicity only. Auto-recovery is not available. Mutually exclusive with CTBEGIN_TRNLOG.</td>
</tr>
<tr>
<td>CTBEGIN_TRNLOG</td>
<td>Full transaction processing functionality including auto-recovery. Mutually exclusive to CTBEGIN_PREIMG. This is the default begin transaction mode.</td>
</tr>
<tr>
<td>CTBEGIN_DEFER</td>
<td>Defer begin transaction until update.</td>
</tr>
<tr>
<td>CTBEGIN_AUTOSAVE</td>
<td>Automatically invokes savepoints after each successful record or resource update.</td>
</tr>
</tbody>
</table>

This mode will be OR-ed in to form the transaction mode used when a c-tree transaction is started. If the transaction mode is CTBEGIN_NONE, the cTRNLOG mode is used to start a new transaction.

Return Values
None.

See Also
CTBase::GetTransactionMode()
**CTBase::SwitchInstance**

Force a c-tree instance switch.

**Declaration**

```cpp
void CTBase::SwitchInstance()
```

**Description**

Force a switch to the c-tree Plus instance indicated by the current session object. Each session object has its unique c-tree instance id.

When most c-treeDB C++ methods are called, they automatically perform a c-tree instance switch. `SwitchInstance()` may be useful before calling specific c-tree ISAM or low level functions to make sure the correct instance is active before making those calls.

**Return**

If any errors are detected, a `CTException` is thrown.

See "c-treeDB Errors and Return Values" (page 881) for a complete listing of valid c-treeDB error codes and return values.

**Example**

```cpp
// declare a remote and local session objects
CTSession hRemote(CTSESSION_CTREE);
CTSession hLocal(CTSESSION_CTREE);

// logon to c-tree server using the remote session object
hRemote.Logon("FAIRCOMS", "ADMIN", "ADMIN");

// logon to local session using the local session object
hLocal.Logon("local", "ADMIN", "ADMIN");

// perform a c-tree instance switch and call //CtreeUserOperation function
hRemote.SwitchInstance();
CtreeUserOperation("!mkdir faircom", buffer, sizeof(buffer));
```

**See also**

`CTRecord::SwitchContext()`, `CTTable::GetDatno()`, `CTTable::GetIdxno()`, `CTIndex::GetIdxno()`
CTBase::Unlock

Syntax
void Unlock ( )

Parameters
This method has no parameters.

Description
Frees all session wide locks held by the program. The session wide locks obtained a call to Lock are also released by Abort() and Commit().

Notice that this call does not free the locks obtained with CTRRecord::LockRecord(). CTRRecord::UnlockRecord() can be used to free just one specific record lock, or CTTable::UnlockTable() can be used to release all records from one table, obtained with CTRRecord::LockRecord().

Return
None.

See also
Lock(), IsLockActive(), GetLockMode(), Commit(), Abort(), CTRRecord::UnlockRecord(), CTTable::UnlockTable()
4.3 CTBigint Class

class CTBigint

Description
The CTBigint class represents big integer objects. Big integer objects are 64-bit integers.

Preconditions
This is one of the basic objects from the c-tree database layer.

CTBigint Operators
- operator =, +=, -=, *=, /=, abs, +, -, *, /: Assignment and arithmetic operators
- operator <, <=, >, >=, ==, !=: Comparison operators
CTBigint::operator=

Syntax
CTBigint& operator=(LONG value)
CTBigint& operator=(CTFLOAT value)
CTBigint& operator=(CTBIGINT value)
CTBigint& operator=(const CTMoney& value)
CTBigint& operator=(const CTString& value)
CTBigint& operator+=(const CTBigint& value)
CTBigint& operator-=(const CTBigint& value)
CTBigint& operator*=(const CTBigint& value)
CTBigint& operator/=(const CTBigint& value)
CTBigint& operator abs(const CTBigint& value)
CTBigint& operator +(const CTBigint& left, const CTBigint& right)
CTBigint& operator -(const CTBigint& left, const CTBigint& right)
CTBigint& operator *(const CTBigint& left, const CTBigint& right)
CTBigint& operator /(const CTBigint& left, const CTBigint& right)

Parameters
- **value**: [in] long, double, CTBIGINT, CTMoney or CTString value or object to be assigned to or updated with the new CTBigint object.
- **left**: [in] The left CTBigint object to be concatenated to form the new CTBigint object
- **right**: [in] The right CTBigint object to be concatenated to form the new CTBigint object

Description
Assigns or concatenates values to form a CTBigint object

Return
The new CTBigint object
CTBigint::operator <, <=, >, >=, ==, !=

Syntax
CTBOOL operator < (const CTBigint& left, const CTBigint& right)
CTBOOL operator <= (const CTBigint& left, const CTBigint& right)
CTBOOL operator > (const CTBigint& left, const CTBigint& right)
CTBOOL operator >= (const CTBigint& left, const CTBigint& right)
CTBOOL operator == (const CTBigint& left, const CTBigint& right)
CTBOOL operator != (const CTBigint& left, const CTBigint& right)

Parameters
- **left**: [in] The left CTBigint object to be compared
- **right**: [in] The right CTBigint object to be compared

Description
These overloaded operators make comparisons between the CTBigint objects.

Return
The operators return YES or NO, depending on the result of the comparison.
CTBigint Methods

Constructor / Destructor
- `CTBigint()`: Creates a `CTBigint` object
- `~CTBigint()`: Destroys a `CTBigint` Object and resets all the dependent objects

Data Handling
- `IsZero()`: Indicate if a big integer value is zero
- `Zero()`: Set a big integer value to zero
- `SetBigint()`: Initializes the big integer object.
- `AsFloat()`: Converts a big integer object to float
- `AsLong()`: Converts a big integer object to long
- `AsBigint()`: Retrieve the C big integer type
- `AsMoney()`: Converts a big integer object to a `CTMoney` object
- `AsString()`: Converts a big integer object to a `CTString` object
CTBigint::CTBigint

Syntax
CTBigint ( )
CTBigint(LONG value)
CTBigint(CTFLOAT value)
CTBigint(CTBIGINT value)
CTBigint(const CTMoney& value)
CTBigint(const CTString& value)
CTBigint(const CTBigint& value)

Parameters
- value: [in] the value to be initially assigned to the new big integer object.

Description
This is the constructor for the CTBigint object.

See also
~CTBigint()
CTBigint::~CTBigint

Syntax
~CTBigint

Parameters
This destructor has no parameters.

Description
This is the destructor for the CTBigint object.

See also
CTBigint()
CTBigInt::AsBigInt

Syntax
CTBigInt AsBigInt( ) const

Parameters
This method has no parameters.

Description
Retrieve the C big integer type.

Return
AsBigInt() returns a CTBigInt object.

See also
AsLong(), AsFloat()
CTBigInt::AsFloat

Syntax
CTFLOAT AsFloat( ) const

Parameters
This method has no parameters.

Description
Converts a CTBigInt object to CTFLOAT.

Return
AsFloat() returns the converted value in CTFLOAT format.

See also
AsLong(), AsBigint()
**CTBigInt::AsLong**

**Syntax**

```cpp
LONG AsLong( ) const
```

**Parameters**

This method has no parameters.

**Description**

Converts a CTBigInt object to LONG.

**Return**

AsLong() returns the converted value in LONG format.

**See also**

AsFloat(), AsBigInt()
CTBigint::AsMoney

Syntax
CTMoney AsMoney( ) const

Parameters
This method has no parameters.

Description
Convert CTBigint (64-bit) to CTMoney (32-bit).

Return
AsMoney() returns the converted value as a CTMoney object.

See also
AsString()
CTBigint::AsString

Syntax
CTStringAsString() const

Parameters
This method has no parameters.

Description
Converts a CTBigint object to a CTString object.

Return
AsString() returns a CTString object with the Bigint() value.

See also
AsBigint()
CTBigint::IsZero

Syntax
CTBOOL IsZero( ) const

Parameters
This method has no parameters.

Description
IsZero() indicates if a big integer value is zero.

Return
IsZero() returns YES if the big integer value is zero, NO otherwise.

See also
Zero()
CTBigint::SetBigint

Syntax
void SetBigint(LONG value)
void SetBigint(CTFLOAT value)
void SetBigint(const CTMoney& value)
void SetBigint(const CTString& value)
void SetBigint(const CTBigint& value)

Parameters
- value [in] The LONG, CTFLOAT, CTBigint, CTMoney, or CTString value or object to be converted and assigned to the new CTBigint object

Description
Initializes the CTBigint object.

Return
None.

See also
Zero()
CTBigint::Zero

Syntax
void Zero( )

Parameters
This method has no parameters.

Description
Zero() sets a Bigint value to zero.

Return
None.

See also
IsZero()
4.4 CTBlob Class

class CTBlob

Description
The `CTBlob` class represents Blob objects.

See Also
`CTString`

Preconditions
This is one of the basic objects from the c-tree database layer.

CTBlob Operators
- operator `=:` Assignment operator
- operator `<`, `<=`, `>`, `>=`, `==`, `!=` Comparison operators
CTBlob::operator =

Syntax
CTBlob& operator=(const pCTBlob pBlob)
CTBlob& operator=(const CTBlob& blob)

Parameters
- **pBlob** [in] The CTBlob object to be assigned to the new CTBlob object
- **blob** [in] The CTBlob object to be assigned to the new CTBlob object

Description
Assigns a Blob or CTBlob object to form a new CTBlob object

Return
The new CTBlob object
CTBlob::operator <, <=, >, >=, ==, !=

Syntax
CTBOOL operator < (const CTBlob& blob)
CTBOOL operator <= (const CTBlob& blob)
CTBOOL operator > (const CTBlob& blob)
CTBOOL operator >= (const CTBlob& blob)
CTBOOL operator == (const CTBlob& blob)
CTBOOL operator != (const CTBlob& blob)

Parameters
- **blob [in]** The CTBlob object to be compared with this CTBlob object.

Description
These overloaded operators compare two CTBlob objects.

Return
The operators return YES or NO, depending on the result of the comparison.
**CTBlob Methods**

**Constructor / Destructor**
- `CTBlob()`: Creates a `CTBlob` object
- `~CTBlob()`: Destroys a `CTBlob` Object and resets all the dependent objects

**Blob Handling**
- `SetBlob()`: Initializes the `CTBlob` object.
- `GetBlob()`: Retrieves a `CTBlob` object
- `Get()`: Retrieves a `CTBlob` object
- `GetSize()`: Retrieves the size of a `CTBlob` object
- `Resize()`: Resizes the `CTBlob` object
- `UpdateBlob()`: Updates the `CTBlob` object.
- `AsString()`: Retrieves the `CTBlob` object as a `CTString` object.
CTBlob::CTBlob

Syntax
CTBlob ( )
CTBlob (const pVOID data, VRLEN size)
CTBlob (const CTString& str)
CTBlob (const pCTBLOB pBlob)
CTBlob (const CTBlob& blob)

Parameters
- data [in] The data to be assigned to the CTBlob object in the creation.
- str [in] The CTString object to be assigned to the new CTBlob object
- pBlob [in] The CTBlob object or Blob to be assigned to the new CTBlob object
- blob [in] The CTBlob object or Blob to be assigned to the new CTBlob object

Description
This is the constructor for the CTBlob object.

See also
- ~CTBlob()
CTBlob::~CTBlob

Syntax
~CTBlob

Parameters
This destructor has no parameters.

Description
This is the destructor for the CTBlob object.

See also
CTBlob()
CTBlob::AsString

Syntax
void AsString(CTString& str) const

Parameters
- `str` [out] the `CTString` object converted from the `CTBlob` object.

Description
Returns a `CTBLOB` object as a `CTString` object.

Return
None.

See also
Get()
CTBlob::Get

Syntax

pCTBLOB Get

Parameters

This method has no parameters.

Description

Retrieves the pCTBLOB member pointer of a CBlob object.

Return

Get returns the CBlob object.

See also

GetBlob(), GetSize()
CTBlob::GetBlob

Syntax

void GetBlob(pCTBLOB pBlob) const
void GetBlob(pVOID& pdata, VRLEN& size)

Parameters

- **pBlob** [out] The retrieved CTBlob object
- **data** [out] The data retrieved from the CTBlob object
- **size** [out] The data size.

Description

Retrieves the CTBlob object

Return

None.

Example

```c
void DisplayDoubles(CTBlob& blob)
{
    DOUBLE* vector;
    VRLEN size;
    int i;

    blob.GetBlob(&vector, &size);
    for (i = 0; i < size / sizeof(DOUBLE); i++)
        printf("%d: %f
", i, vector[i];

    _ctdb_free(vector);
}
```

See also

SetBlob(), Get()
CTBlob::GetSize

Syntax
VRLENGetSize const

Parameters
This method has no parameters.

Description
Retrieves the CTBlob object size.

Return
GetSize() returns the object size.

See also
ResizeObject()
CTBlob::Resize

Syntax
void Resize(VRLEN size)

Parameters
- size [in] The new value to be assigned to the CTBlob object

Description
Resizes the CTBlob object.

Return
None.

See also
GetSize()
**CTBlob::SetBlob**

**shouSyntax**
void SetBlob(const pCTBLOB pBlob)
void SetBlob(const CTblob& blob)
void SetBlob(const pVOID data, VRLEN size)
void SetBlob(const CTString& str)

**Parameters**
- **pBlob** [in] The *CTBlob* object or Blob to be assigned to the *CTBlob* object
- **blob** [in] The *CTBlob* object or Blob to be assigned to the *CTBlob* object
- **data** [in] The data to be assigned to the *CTBlob* object
- **size** [in] The data size.
- **str** [in] The *CTString* object to be assigned to the *CTBlob* object

**Description**
Initializes the *CTBlob* object.

**Return**
None.

**See also**
*GetBlob()*, *Get()*
CTBlob::UpdateBlob

Syntax

```cpp
void UpdateBlob(const pVOID data, VRLEN size, VRLEN offset)
```

Parameters

- `data` [in] The new data to be assigned to the `CTBlob` object
- `offset` [in] The initial offset in the `CTBlob` object to start updating.

Description

Updates one existing `CTBlob` object.

Return

None.

See also

`GetBlob()`
4.5 CTCurrency Class

class CTCurrency

Description
The CTCurrency class represents Currency objects. Currency objects are 64-bit integers representing currency values.

See Also
CTMoney, CTString

Preconditions
This is one of the basic objects from the c-tree database layer.

CTCurrency Operators
- operator =, +=, -=, *=, /=, abs, +, -, *, /: Assignment and arithmetic operators
- operator <, <=, >, >=, ==, !=: Comparison operators
**Syntax**

CTCurrency& operator=(LONG value)
CTCurrency& operator=(CTFLOAT value)
CTCurrency& operator=(CTCURRENCY value)
CTCurrency& operator=(const CTMoney& value)
CTCurrency& operator=(const CTString& value)
CTCurrency& operator=(const CTBigint& value)
CTCurrency& operator=(const CTCurrency& value)
CTCurrency& operator+=(const CTCurrency& value)
CTCurrency& operator-=(const CTCurrency& value)
CTCurrency& operator*=(const CTCurrency& value)
CTCurrency& operator/=(const CTCurrency& value)
CTCurrency& operator abs(const CTCurrency& value)
CTCurrency& operator +(const CTCurrency& left, const CTCurrency& right)
CTCurrency& operator -(const CTCurrency& left, const CTCurrency& right)
CTCurrency& operator *(const CTCurrency& left, const CTCurrency& right)
CTCurrency& operator /(const CTCurrency& left, const CTCurrency& right)

**Parameters**

- **value** [in] LONG, double, CTCurrency, CTMoney, CTString, CTBigint or CTCurrency value or object to be assigned to or updated with the new CTCurrency object.
- **left** [in] The left CTCurrency object to be concatenated to form the new CTCurrency object
- **right** [in] The right CTCurrency object to be concatenated to form the new CTCurrency object

**Description**

Assigns or concatenates values to form a CTCurrency object

**Return**

The new CTCurrency object
CTCurrency::operator <, <=, >, >=, ==, !=

Syntax
CTBOOL operator < (const CTCurrency& left, const CTCurrency& right)
CTBOOL operator <= (const CTCurrency& left, const CTCurrency& right)
CTBOOL operator > (const CTCurrency& left, const CTCurrency& right)
CTBOOL operator >= (const CTCurrency& left, const CTCurrency& right)
CTBOOL operator == (const CTCurrency& left, const CTCurrency& right)
CTBOOL operator != (const CTCurrency& left, const CTCurrency& right)

Parameters
- left [in] The left CTCurrency object to be compared
- right [in] The right CTCurrency object to be compared

Description
These overloaded operators make comparisons between two CTCurrency objects.

Return
The operators return YES or NO, depending on the result of the comparison.
CTCurrency Methods

Constructor / Destructor
- `CTCurrency()`: Creates a `CTCurrency` object
- `~CTCurrency()`: Destroys a `CTCurrency` Object and resets all the dependent objects

Currency Handling
- `IsZero()`: Indicates if a currency value is zero
- `Zero()`: Sets a currency value to zero
- `SetCurrency()`: Initializes the `CTCurrency` object.
- `AsFloat()`: Converts a `CTCurrency` object to float
- `AsLong()`: Converts a `CTCurrency` object to long
- `AsBigInt()`: Converts a `CTCurrency` object to big integer
- `AsCurrency()`: Retrieves the currency type
- `AsString()`: Converts a `CTCurrency` object to a `CTString` object
- `StringToCurrency()`: Converts a `CTString` object to a `CTCurrency` object
CTCurrency::CTCurrency

Syntax
CTCurrency ( )
CTCurrency (LONG value)
CTCurrency (CTFLOAT value)
CTCurrency (CTCURRENCY value)
CTCurrency(const CTMoney& value)
CTCurrency(const CTString& value)
CTCurrency(const CTBigint& value)
CTCurrency(const CTCurrency& value)

Parameters
- value: [in] The LONG, double, CTCurrency, CTMoney, CTString, CTBigint or CTCurrency value or object to be converted and assigned to the new CTCurrency object

Description
This is the constructor for the CTCurrency object.

See also
~CTCurrency()
CTCurrency::~CTCurrency

Syntax
~CTCurrency

Parameters
This destructor has no parameters.

Description
This is the destructor for the CTCurrency object.

See also
CTCurrency()
CTCurrency::AsBigInt

Syntax
CTBigInt AsBigInt( ) const

Parameters
This method has no parameters.

Description
Convert a CTCurrency object to big integer.

Return
AsBigInt() returns the converted value in big integer (64-bit) format.

See also
AsLong(), AsFloat()
CTCurrency::AsCurrency

Syntax
CTCURRENCY AsCurrency( ) const

Parameters
This method has no parameters.

Description
Retrieve the C currency type.

Return
AsCurrency() returns a CTCURRENCY type.

See also
AsString()
**CTCurrency::AsFloat**

**Syntax**

```cpp
CTFLOAT AsFloat  const
```

**Parameters**

This method has no parameters.

**Description**

Converts a `CTCurrency` object to `CTFLOAT`.

**Return**

`AsFloat()` returns the converted value in `CTFLOAT` format.

**See also**

`AsLong()`, `AsBigint()`
CTCurrency::AsLong

Syntax
LONG AsLong( ) const

Parameters
This method has no parameters.

Description
Converts a CTCurrency object to LONG.

Return
AsLong() returns the converted value in LONG format.

See also
AsFloat(), AsBigInt()
CTCurrency::AsMoney

Syntax
CTMoney AsMoney( ) const

Parameters
This method has no parameters.

Description
Convert a CTCurrency object (64-bit) to a CTMoney object (32-bit).

Return
AsMoney() returns the converted value as a CTMoney object.

See also
AsString()
CTCurrency::AsString

Syntax
CTStringAsString( ) const

Parameters
This method has no parameters.

Description
Converts a CTCurrency object to a CTString object.

Return
AsString() returns a CTString object with the Currency value.

See also
StringToCurrency(), AsCurrency()
CTCurrency::IsZero

Syntax
CTBOOL IsZero( ) const

Parameters
This method has no parameters.

Description
IsZero() indicates if a currency value is zero.

Return
IsZero() returns YES if the currency value is zero, NO otherwise.

See also
Zero()
CTCurrency::Round

Syntax
CTCurrency Round (NINT scale) const

Parameters
- scale [in] the number of decimal places.

Description
Rounds a CTCurrency to a specified number of decimal places.

Return
Round returns the rounded CTCurrency.

See also
AsString(), SetCurrency()
CTCurrency::SetCurrency

Syntax
void SetCurrency(LONG value)
void SetCurrency(CTFLOAT value)
void SetCurrency(const CTMoney& value)
void SetCurrency(const CTString& value)
void SetCurrency(const CTBigint& value)

Parameters
- value [in] The LONG, double, CTCurrency, CTMoney, CTString, CTBigint or CTCurrency value or object to be converted and assigned to the CTCurrency object

Description
Initializes the CTCurrency object.

Return
None.

See also
Zero()
CTCurrency::StringToCurrency

Syntax
void StringToCurrency(const CTString& str)

Parameters
- str [in] The string object to be converted.

Description
StringToCurrency() converts a CTString object to a CTCurrency object.

Return
None.

See also
AsString()
**CTCurrency::Zero**

**Syntax**
```cpp
void Zero ( )
```

**Parameters**
This method has no parameters.

**Description**

Zero() sets a CTCurrency object value to zero.

**Return**
None.

**See also**
IsZero()
4.6 CTDatabase Class

class CTDatabase

Description
The CTDatabase class deals with the Database concept. The Database can be thought as a collection of tables. Multiple databases can be connected at the same time, within the same session. The same database may open at different sessions. CTDatabase class is derived from the CTBase class.

See Also
CTBase, CTSession, CTTable

Preconditions
Before performing any tasks with a CTDatabase object, it is necessary to initialize one CTSession object.
CTDatabase Methods

Constructor / Destructor
- **CTDatabase()**: Creates a *CTDatabase* object
- **~CTDatabase()**: Destroys a *CTDatabase* Object and resets all the dependent objects

Database Handling
- **IsActive()**: Retrieves the active state of a database (connected or disconnected state).
- **Connect()**: Connects to a database
- **Create()**: Create a new database
- **Disconnect()**: Disconnects from a specific database
- **GetName()**: Retrieves the database name.
- **GetPath()**: Retrieves the database path

Table Handling
- **FirstTable()**: Locates the first table in a database.
- **NextTable()**: Locates the next table in a database.
- **FindTable()**: Locates a table in a database by name.
- **AddTable()**: Adds an existing table to a database.
- **DeleteTable()**: Drops a table from the database, and delete the files from disk.
- **DropTable()**: Drops a table from the database, but does not delete the files from disk.
- **GetTableUID()**: Retrieves the table UID.
- **GetFirstActive()**: Retrieves the first active table in the database.
- **GetNextActive()**: Retrieves the next active table in the database.
- **FindActive()**: Retrieves active table
- **CloseAll()**: Closes all active tables associated with the database.
- **GetTableCount()**: Retrieves the number of tables in the database dictionary.
CTDatabase::CTDatabase

Syntax
CTDatabase(const CTSession& Handle)
CTDatabase(const CTSession* Handle)

Parameters
- Handle [in] the CTSession object, or a pointer to the CTSession object, to which the
  CTDDatabase object is to be created or connected.

Description
This is the constructor for the CTDDatabase class.

See also
~CTDatabase(), CTSession()
CTDatabase::~CTDatabase

Syntax
~CTDatabase

Parameters
This destructor has no parameters.

Description
This is the destructor for the CTDatabase class. It frees the allocated memory to the Database. No database operations are allowed after the CTDatabase destructor is called.

See also
CTDatabase()
CTDatabase::AddTable

Syntax
void AddTable(const CTString& Name, const CTString& Path)

Parameters
- **Name** [in] The name of the table to add to the database.
- **Path** [in] The table path to add to the database.

Description
Adds an existing table to the database. It is possible to add c-tree Plus files to an existing database if this file has the incremental and DODA structures stored. See "Compatibility" (page 220) regarding compatibility issues.

A table that has been created independent from a database, or in a different database may also be added to the current database.

Return
None.

Example

```cpp
pDatabase->Connect();
try
{
    pDatabase->AddTable(my_old_c-tree_file, path);
    printf("\nTable added successfully!!\n");
}
catch (CTException& err)
{
    printf("\n\n%d  %s", err.GetErrorCode(),
            err.GetErrorMsg());
    return (1);
}
```

See also
DropTable(), DeleteTable(), CreateTable()
CTDatabase::CloseAll

Syntax

void CloseAll();

Parameters

This method has no parameters.

Description

Closes all active tables associated with the database. To close one specific table, the method CTTable::Close() may be used.

Return

None.

Example

pDatabase->CloseAll();
pSession->DisconnectAll();
pSession->Logout();

See also

CTTable::Close()
CTDatabase::Connect

Syntax
void Connect(const CTString& Name)

Parameters
- Name [in] The database name to be connected.

Description
Connects to a database

Return
None.

Example

pSession = new CTSesion;
pDatabase = new CTDatabase(pSession);
pSession->Logon();
pDatabase->Connect("myDatabase");

See also
Disconnect()
CTDatabase::Create

Syntax
void Create(const CTString& Name, const CTString& Path)

Parameters
- **Name** [in] The database name to be created.
- **Path** [in] The path where the database will be created.

Description
Creates a new database

Return
None.

See also
CTSession::CreateDatabase()
CTDatabase::DeleteTable

Syntax
void DeleteTable(const CTString& Name, const CTString& Password)

Parameters
- **Name** [in] The name of the table to delete from the database.
- **Password** [in] The table password.

Description
Drops a table from the database, and delete the files from disk.

Return
None.

Example
pMyTable->Close(); // close the custmast table.
pMyDatabase->DeleteTable("custmast", my_table_password);

See also
DropTable(), AddTable()
CTDatabase::Disconnect

Syntax
void Disconnect()

Parameters
This method has no parameters.

Description
Disconnects a specific database.

Return
None.

Example
pDatabase->Connect();
pDatabase->FirstTable(tbname, tbpath);
do
  { printf("\ntable name: %s",tbname.c_str()); }  
while (NextTable(tbname, tbpath);
  pDatabase->Disconnect();

See also
Connect()
CTDatabase::DropTable

Syntax

void DropTable(const CTString& Name)

Parameters

- Name [in] The name of the table to drop from the database.

Description

Drops a table from the database, but does not delete the files from disk. After one table has been dropped from the database, it may be operated independent from a Database using the CTSession::SetLogonOnly() method, or it may be added to any other database in the system.

Return

None.

Example

pMyTable->Close(); // close the custmast table.
pMyDatabase->DropTable("custmast");

See also

DeleteTable(), AddTable(), CTSession::SetLogonOnly()
CTDatabase::FindActive

Syntax
CTTable* FindActive(const CTString& Name)
CTTable* FindActive(ULONG uid)

Parameters
- **Name** [in] The table name
- **uid** [in] The table uid

Description
Retrieves active table. If the prototype `FindActive(const CTString& Name)` is used, the table name is searched. If the prototype `FindActive(const CTString& Name)` is used, the table uid is searched.

Return
`FindActive()` returns the table object handle, or null if the table no exists or is not active.

See also
`FirstActive()`, `IsActive()`
CTDatabase::FindTable

Syntax

CTBOOL FindTable(const CTString& Name, CTString& Path)
CTBOOL FindTable(const CTString& Name, CTString& Path, ULONG& uid)
CTBOOL FindTable(ULONG uid, CTString& Name, CTString& Path)

Parameters

- **Name** [in or out] The name of the table. When using the `FindTable(ULONG uid, CTString& Name, CTString& Path)` prototype, `Name` is an output parameter. Otherwise, it is an input, and it is used to locate the table.
- **Path** [out] The table path, and it is returned by the method.
- **uid** [in or out] The table UID. When using the `FindTable(const CTString& Name, CTString& Path, ULONG& uid)` prototype, `uid` is an output parameter. When using the `FindTable(ULONG uid, CTString& Name, CTString& Path)` prototype, `uid` is an input parameter.

Description

Locates a table in a database. If the prototype `FindTable(const CTString& Name, CTString& Path)` is used, the table is searched by name, and the method retrieves its path. If the prototype `FindTable(const CTString& Name, CTString& Path, ULONG& uid)` is used, the table is searched by name, and the method retrieves its path and uid. If the prototype `FindTable(ULONG uid, CTString& Name, CTString& Path)` is used, the table is searched by uid, and the method retrieves its name and path.

Return

`FindTable()` returns YES if the table is found, NO otherwise.

See also

FirstTable()
CTDatabase::FirstTable

Syntax
CTBOOL FirstTable(CTString& Name, CTString& Path)

Parameters
- Name [out] The name of the first table in the database
- Path [out] The path of the first table in the database.

Description
Locates the first table in a database.

Return
FirstTable() returns YES if one table is found, NO otherwise.

Example
pDatabase->Connect();
pDatabase->FirstTable(tname, tbpath);
do
    { printf("\n\table name: %s", tname.c_str()); }
while (NextTable(tname, tbpath);

See also
NextDaTable(), FindTable(), GetFirstActive()
CTDatabase::GetFirstActive

Syntax
CTTable* GetFirstActive( )
CTTable* GetFirstActive(VRLEN& ScanRec)

Parameters
- **ScanRec** [out] Holds the state of the first or next scan, and is returned by the method if the prototype `GetFirstActive(VRLEN& ScanRec)` is used.

Description
Retrieves the first active table in the database. One table is active if it is open. If more than one scan must be done at the same time, `GetFirstActive(VRLEN& ScanRec)` must be used, with different `ScanRec` parameters for each scan.

Return
`GetFirstActive()` returns the table object pointer.

Example
```cpp
pTable=pDatabase->GetFirstActive();
printf("\nFirst Active table: %s",
       pTable->GetName().c_str());
```

See also
- `GetNextActive()`, `FirstTable()`, `IsActive()`
CTDatabase::GetNextActive

Syntax
CTTable* GetNextActive
CTTable* GetNextActive(VRLEN& ScanRec)

Parameters
- ScanRec [in and out] Holds the state of the first or next scan. It is an input to the method, and it is at the same time, an output from the method when the prototype GetNextActive(VRLEN& ScanRec) is used.

Description
Retrieves the next active table in the database. One table is active if it is open. If more than one scan must be done at the same time, GetNextActive(VRLEN& ScanRec) must be used, with different ScanRec parameters for each scan.

Return
GetNextActive() returns the table object pointer.

See also
GetFirstActive(), IsActive()
**CTDatabase::GetName**

**Syntax**

```cpp
CTString GetName( )
```

**Parameters**

This method has no parameters.

**Description**

Retrieves the database name.

**Return**

- `GetName()` returns the database name.

**See also**

- `GetPath()`
CTDatabase::GetPath

Syntax
CTString GetPath( )

Parameters
This method has no parameters.

Description
Retrieves the database path.

Return
GetPath() returns the database path.

See also
GetName()
CTDatabase::GetTableCount

Syntax
NINT GetTableCount();

Parameters
This method has no parameters.

Description
Retrieves the number of tables in the database dictionary.

Return
GetTableCount() returns the number of tables in the database.

Example

pDatabase->Connect();
n=pDatabase->GetTableCount();
printf("\nThere are \d tables in this database.",n);
CTDatabase::GetTableUID

Syntax
void GetTableUID(const CTString& Name, ULONG& uid)

Parameters
- **Name** [in] The table name, given to the method.
- **uid** [out] The table UID, returned by the method.

Description
Retrieves the table UID.

Return
None.

See also
SetTableUID()
CTDatabase::IsActive

Syntax
CTBOOL IsActive()

Parameters
This method has no parameters.

Description
Retrieves the active state of a database (connected or disconnected state).

Return
IsActive() returns YES if the database is connected, or NO if database is disconnected.

Example
if (!pDatabase->IsActive())
    pDatabase->Connect();

See also
FindActive()
CTDatabase::IsExclusive

Retrieves the status of the database exclusive flag.

Syntax

CTBOOL CTDatabase::IsExclusive() const;

Description

CTDatabase::IsExclusive() retrieves the status of the database exclusive flag.

Return

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO</td>
<td>Database is shared.</td>
</tr>
<tr>
<td>1</td>
<td>YES</td>
<td>Database is exclusive.</td>
</tr>
</tbody>
</table>

Example

```c
// verify an exclusive logon and connect
CTSession hSession(CTSESSION_CTDB);
CTDatabase hDatabase(hSession);

hSession.SetSessionExclusive(YES);
hSession.Logon("FAIRCOMS", "ADMIN", "ADMIN");
if (hSession.IsExclusive())
    printf("Session is exclusive\n");
else
    printf("Session is shared\n");

hDatabase.SetExclusive(YES);
hDatabase.Connect("MyData");
if (hDatabase.IsExclusive())
    printf("Database is exclusive\n");
else
    printf("Database is shared\n");
```

See Also

CTSession::SetExclusive(), CTSession::IsExclusive(), CTDatabase::SetExclusive()
**Syntax**

CTBOOL NextTable(CTString& Name, CTString& Path)

**Parameters**

- *Name* [out] The name of the next table in the database.
- *Path* [out] The path of the next table in the database.

**Description**

Locates the next table in a database. `FirstTable()` must be used before using `NextTable()` for the first time in a database.

**Return**

`NextTable()` returns YES if one table is found, NO otherwise.

**Example**

```c
pDatabase->Connect();
pDatabase->FirstTable(tbname, tbpath);
do
  { printf("table name: %s",tbname.c_str()); }
while (NextTable(tbname, tbpath));
```

**See also**

`FirstTable()`
CTDatabase::RenameTable

Declaration

CTDatabase::RenameTable(const CTString& OldName, const CTString& NewName);

Description

- **OldName**  a string containing the original table name
- **NewName**  a string containing the new table name

Return Values

None. Throws a `CTException` if an error occurs.

See Also
**CTDatabase::SetExclusive**

Sets or clears the database exclusive flag.

**Declaration**

```c
void CDatabase::SetExclusive(CTBOOL flag);
```

**Description**

*flag* will set or clear the database exclusive flag.

**CTDatabase::SetExclusive()** sets or clears the database exclusive flag. If a database exclusive flag is set, only one connection will be allowed on this database. Set the database exclusive flag after allocating the database handle, but before performing a connect. Setting the database exclusive flag after a database is connected will not have any effect during the current connection.

**Return**

*void*

**Example**

```c
// perform an exclusive logon and connect
CTSession hSession(CTSESSION_CTDB);
CTDatabase hDatabase(hSession);

hSession.SetExclusive(YES);
hSession.Logon(“FAIRCOMS”, “ADMIN”, “ADMIN”);
hDatabase.SetExclusive(YES);
hDatabase.Connect(“MyData”);
```

**See Also**

*CTSession::SetExclusive(), CTSession::IsExclusive(), CTDatabase::IsExclusive()*
4.7 CDate Class

class CDate

Description
The CDate class represents Date objects.

See Also
CTDateTime(), CTTime()

Preconditions
This is one of the basic objects from the c-tree database layer.

CDate Operators
- operator = : Assignment operator
- operator <, <=, >, >=, ==, != : Comparison operators
CTDate::operator =

Syntax
CTDate& operator = (const CTDate& Date)
CTDate& operator = (const CTDATE Date)

Parameters
- Date [in] The CDate object or date to be assigned to the new CDate object

Description
These overloaded operators assign a date or CDate object to a CDate object

Return
The new CDate object
CTDate::operator <, <=, >, >=, ==, !=

Syntax

CTBOOL operator < (const CTDate& Date)
CTBOOL operator <= (const CTDate& Date)
CTBOOL operator > (const CTDate& Date)
CTBOOL operator >= (const CTDate& Date)
CTBOOL operator == (const CTDate& Date)
CTBOOL operator != (const CTDate& Date)

Parameters

- *Date* [in] The date to be compared to this *CTDate* object

Description

These overloaded operators make comparisons between the *CTDate* object and *Date*.

Return

The operators return YES or NO, depending on the result of the comparison.
CTDate Methods

Constructor / Destructor
- **CTDate()**: Creates a *CTDate* object
- **~CTDate()**: Destroys a *CTDate* Object and resets all the dependent objects

Date Handling
- **GetDate()**: Retrieves the date.
- **SetDate()**: Sets the date to the *CTDate* object.
- **Pack()**: Packs a year, month and day value into a *CTDATE* type value.
- **Unpack()**: Unpacks a *CTDATE* type value into a year, month and day values
- **Year()**: Retrieves the year component of a packed *CTDATE* type value
- **Month()**: Retrieves the month component of a packed *CTDATE* type value
- **Day()**: Retrieves the day component of a packed *CTDATE* type value
- **DayOfWeek()**: Retrieves the day of the week from a packed *CTDATE* type value.
- **IsLeapYear()**: Indicates if the year component of a packed *CTDATE* type value is a leap year
- **DateToString()**: Converts a packed *CTDATE* type value to a *CTString* object
- **StringToDate()**: Converts a *CTString* object to a date object
- **CurrentDate()**: Retrieves the current system date.
CTDate::CTDate

Syntax
CTDate( )
CTDate(NINT year, NINT month, NINT day)
CTDate(CTDATE Date)
CTDate(const CTDate& Date)

Parameters
- year [in] The year to set to the CTDDate object.
- month [in] The month to set to the CTDDate object.
- day [in] The day to set to the CTDDate object.
- Date [in] The date to set to the CTDDate object.

Description
This is the constructor for the CTDDate object.

See also
- ~CTDate()
CTDate::~CTDate

Syntax
~CTDate

Parameters
This destructor has no parameters.

Description
This is the destructor for the CTDDate object.

See also
CTDate()
CTDate::CurrentDate

Syntax
CTDate CurrentDate()

Parameters
This method has no parameters.

Description
Retrieves the current system date

Return
CurrentDate() returns a CDate object with the system current date.

See also
GetDate()
CTDate::DateToString

Syntax

```cpp
CTString DateToString(CTDATE_TYPE DateType) const
```

Parameters

- **DateType** [in] The date type format used to convert to string. The valid date type formats are:

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTDATE_DEF</td>
<td>Use default date type</td>
</tr>
<tr>
<td>CTDATE_MDCY</td>
<td>Date is mm/dd/ccyy</td>
</tr>
<tr>
<td>CTDATE_MDY</td>
<td>Date is mm/dd/yy</td>
</tr>
<tr>
<td>CTDATE_DMCY</td>
<td>Date is dd/mm/ccyy</td>
</tr>
<tr>
<td>CTDATE_DMY</td>
<td>Date is dd/mm/yy</td>
</tr>
<tr>
<td>CTDATE_CYMD</td>
<td>Date is ccyymmd</td>
</tr>
<tr>
<td>CTDATE_YMD</td>
<td>Date is yymmd</td>
</tr>
</tbody>
</table>

Description

Converts a packed `CTDATE` type value to a `CTString` object.

Return

`DateToString()` returns a `CTString` object with the date.

See also

- `StringToDate()`
CTDate::Day

Syntax
NINT Day( ) const

Parameters
This method has no parameters.

Description
Retrieves the day component of a packed CTDATE type value.

Return
Day returns the unpacked day.

See also
Month(), Year(), DayOfWeek(), Unpack(), Pack()
CTDate::DayOfWeek

Syntax
NINT DayOfWeek() const

Parameters
This method has no parameters.

Description
Retrieves the day of the week from a packed CTDATE type value. 0 is Sunday, 1 is Monday, ..., 6 is Saturday.

Return
DayOfWeek() returns the day of the week.

See also
Day()
CTDate::GetDate

Syntax

CTDATE GetDate( ) const

Parameters

This method has no parameters.

Description

Retrieves the date.

Return

GetDate() returns a CTDATE object with the date.

See also

SetDate(), CurrentDate()
CTDate::IsLeapYear

Syntax
CTBOOL IsLeapYear( ) const

Parameters
This method has no parameters.

Description
Indicates if the year component of a packed CTDATE type value is a leap year.

Return
IsLeapYear() returns YES if the packed year is a leap year, and NO otherwise.

See also
Year()
CTDate::Month

Syntax
NINT Month( ) const

Parameters
This method has no parameters.

Description
Retrieves the month component of a packed CTDATE type value

Return
Month() returns the unpacked month.

See also
Unpack(), Year(), Day(), Pack()
CTDate::Pack

Syntax
void Pack(NINT year, NINT month, NINT day)

Parameters
- *year* [in] The year to pack.
- *day* [in] The day to pack.

Description
Packs a year, month and day value into a CDATE type value.

Return
None.

See also
Unpack()
CTDate::SetDate

Syntax

void SetDate(CTDATE Date)
void SetDate(const CTDate& Date)

Parameters

- Date [in] The new date to set to the CTDate object.

Description

Sets the date to the CTDate object.

Return

None.

See also

GetDate()
**CTDate::StringToDate**

**Syntax**

```cpp
void StringToDate(const CTString& str, CTDATA_TYPE DateType)
```

**Parameters**

- `str` [in] The string object to be converted.
- `DateType` [in] The date type format to be used in the conversion. Valid date formats are shown in "Data Types" (page 163).

**Description**

Converts a `CTString` object to a date object.

**Return**

None.

**See also**

`DateToString()`
CTDate::Unpack

Syntax
void Unpack(NINT& year, NINT& month, NINT& day) const

Parameters
- `year` [out] The unpacked year.
- `month` [out] The unpacked month.
- `day` [out] The unpacked day.

Description
Unpacks a `CTDATE` type value into a year, month and day values

Return
None.

See also
Pack(), Year(), Month(), Day(), DayOfWeek()
CTDate::Year

Syntax
NINT Year() const

Parameters
This method has no parameters.

Description
Retrieves the year component of a packed CTDATE type value.

Return
Year() returns the unpacked year.

See also
Month(), Day(), Unpack(), Pack()
4.8 **CTDateTime Class**

**class CTDateTime**

**Description**
The `CTDateTime` class represents `DateTime` objects.

**See Also**
`CTDateTime` class, `CTDateTime` class

**Preconditions**
This is one of the basic objects from the c-tree database layer.

**CTDateTime Operators**
- operator `=`: Assignment operator
- operator `<`, `<=`, `>`, `>=`, `==`, `!=`: Comparison operators
CTDateTime::operator =

Syntax
CTDateTime& operator = (CTDATETIME DateTime)
CTDateTime& operator = (const CTDateTime& DateTime)

Parameters
- DateTime [in] The CDateTime object or DateTime to be assigned to the new CDateTime object

Description
These overloaded operators assign a DateTime or CDateTime object to a CDateTime object

Return
The new CDateTime object
**Syntax**
CTBOOL operator < (const CTDateTime& DateTime)
CTBOOL operator <= (const CTDateTime& DateTime)
CTBOOL operator > (const CTDateTime& DateTime)
CTBOOL operator >= (const CTDateTime& DateTime)
CTBOOL operator == (const CTDateTime& DateTime)
CTBOOL operator != (const CTDateTime& DateTime)

**Parameters**
- **DateTime** [in] The **DateTime** to be compared to this **CTDateTime** object

**Description**
These overloaded operators make comparisons between the **CTDateTime** object and **DateTime**.

**Return**
The operators return YES or NO, depending on the result of the comparison.
CTDateTime Methods

Constructor / Destructor
- **CTDateTime()**: Creates a `CTDateTime` object
- **~CTDateTime()**: Destroys a `CTDateTime` Object and resets all the dependent objects

DateTime Handling
- **GetDateTime()**: Retrieves the `DateTime`
- **SetDateTime()**: Sets the `DateTime` to the `CTDateTime` object.
- **Pack()**: Packs an hour, minute and second value into a `CTDATETIME` type value.
- **Unpack()**: Unpacks a `CTDATETIME` type value into hour, minute and second values
- **GetDate()**: Retrieves a `CTDate` object from a `CTDateTime` object.
- **GetTime()**: Retrieves a `CTTime` object from a `CTDateTime` object.
- **Year()**: Retrieves the year component of a `CTDATETIME` object.
- **Month()**: Retrieves the month component of a `CTDATETIME` object.
- **Day()**: Retrieves the day component of a `CTDATETIME` object.
- **DayOfWeek()**: Retrieves the day of the week from the date part of the `CTDATETIME` object.
- **IsLeapYear()**: Indicates if the year component from a `CTDATETIME` object is a leap year.
- **Hour()**: Retrieves the hour component from a `CTDATETIME` type value
- **Minute()**: Retrieves the minute component from a `CTDATETIME` type value
- **Second()**: Retrieves the second component from a `CTDATETIME` type value
- **DateTimeToString()**: Converts a packed `CTDATETIME` type value to a `CTString` object
- **StringToDateTime()**: Converts a `CTString` object to a `DateTime` object
- **CurrentDateTime()**: Retrieves the current system date and time.
- **CurrentDate()**: Retrieve the current system date and sets the time to zeros
CTDateTime::CTDateTime

Syntax
CTDateTime();
CTDateTime(NINT year, NINT month, NINT day, NINT hour = 0, NINT minute = 0, NINT second = 0);
CTDateTime(CTDATE Date, CTTIME Time = 0);
CTDateTime(const CTDate& Date, const CTTime& Time = 0);
CTDateTime(CTDATETIME DateTime);
CTDateTime(CTDateTime& DateTime);

Parameters
- **year** [in] The year to set to the CTDate object.
- **month** [in] The month to set to the CTDate object.
- **day** [in] The day to set to the CTDate object.
- **hour** [in] The hour to set to the CTDateTime object.
- **minute** [in] The minute to set to the CTDateTime object.
- **second** [in] The second to set to the CTDateTime object.
- **Date** [in] The date to set to the CTDateTime object.
- **Time** [in] The Time to set to the CTDateTime object.
- **DateTime** [in] The DateTime to set to the CTDateTime object.

Description
This is the constructor for the CTDateTime object.

See also
~CTDateTime()
CTDateTime::~CTDateTime

Syntax
~CTDateTime

Parameters
This destructor has no parameters.

Description
This is the destructor for the CDateTime object.

See also
CTDateTime()
CTDateTime::CurrentDate

Syntax
CTDate CurrentDate( )

Parameters
This method has no parameters.

Description
Retrieves the current system date and sets the time to zero.

Return
CurrentDate() returns a CDate object with the system current date.

See also
GetDate(), CurrentDateTime()
CTDateTime::CurrentDateTime

Syntax
CTDateTime CurrentDateTime()

Parameters
This method has no parameters.

Description
Retrieves the current system date and time

Return
CurrentDateTime() returns a CTDateTime object with the system current DateTime.

See also
GetDateTime(), CurrentDate()
**CTDateTime::DateTimeToString**

**Syntax**

```cpp
CTString DateTimeToString(CTDATE_TYPE DateType, CTTIME_TYPE TimeType) const
```

**Parameters**

- *DateType* [in] The Date type format used to convert to string. The valid Date type formats are shown in "Data Types" (page 163).
- *TimeType* [in] The Time type format used to convert to string. The valid Time type formats are shown in "Data Types" (page 163).

**Description**

Converts a packed `CTDATETIME` type value to a `CTString` object.

**Return**

`DateTimeToString()` returns a `CTString` object with the `DateTime`.

**See also**

`StringToDate`
**Syntax**

NINT Day( ) const

**Parameters**

This method has no parameters.

**Description**

Retrieves the day component from a `CTDATETIME` object.

**Return**

`Day()` returns the unpacked day.

**See also**

`Month()`, `Year()`, `DayOfWeek()`, `Unpack()`, `Pack()`
### CTDATETIME::DayOfWeek

**Syntax**

```cpp
NINT DayOfWeek() const
```

**Parameters**

This method has no parameters.

**Description**

Retrieves the day of the week from the date part of the `CTDATETIME` object. 0 is Sunday, 1 is Monday, ..., 6 is Saturday.

**Return**

`DayOfWeek()` returns the day of the week.

**See also**

`Day()`
CTDateTime::GetDate

Syntax
CTDATE GetDate( ) const

Parameters
This method has no parameters.

Description
Retrieves a CDate object from a CDateTime object.

Return
GetDate() returns a CDATE object with the date.

See also
SetDate(), GetDateTime()
CTDateTime::GetDateTime

Syntax
CTDATETIME GetDateTime( ) const

Parameters
This method has no parameters.

Description
Retrieves the DateTime.

Return
GetDateTime() returns a CTDATETIME object with the DateTime.

See also
SetDateTime(), CurrentDateTime()
CTDateTime::GetTime

Syntax
CTTIME GetTime( ) const

Parameters
This method has no parameters.

Description
Retrieves a CTime object from a CDateTime object

Return
GetTime() returns a CTIME object with the Time.

See also
SetTime(), GetDateTime()
**CTDateTime::IsLeapYear**

**Syntax**

```cpp
CTBOOL IsLeapYear() const
```

**Parameters**

This method has no parameters.

**Description**

Indicates if the year component from a `CTDateTime` object is a leap year.

**Return**

`IsLeapYear()` returns YES if the year is a leap year and NO otherwise.

**See also**

`Year()`
**CTDateTime::Hour**

**Syntax**

```
NINT Hour() const
```

**Parameters**

This method has no parameters.

**Description**

Retrieves the hour component from a `CTDATETIME` object.

**Return**

`Hour()` returns the unpacked hour.

**See also**

`Minute()`, `Second()`, `Unpack()`, `Pack()`
CTDateTime::Minute

Syntax
NINT Minute( ) const

Parameters
This method has no parameters.

Description
Retrieves the minute component from a CTDATETIME object

Return
Minute() returns the unpacked minute.

See also
Unpack(), Hour(), Second(), Pack()
CTDateTime::Month

Syntax
NINT Month( ) const

Parameters
This method has no parameters.

Description
Retrieves the month component from a CTDATETIME object

Return
Month( ) returns the unpacked month.

See also
Unpack(), Year(), Day(), Pack()
CTDateTime::Pack

Syntax
void Pack(NINT year, NINT month, NINT day, NINT hour = 0,
NINT minute = 0, NINT second = 0)

Parameters
- year [in] The year to pack.
- day [in] The day to pack.
- hour [in] The hour to pack.
- minute [in] The minute to pack.
- second [in] The second to pack.

Description
Packs an hour, minute and second value into a CTDATETIME object.

Return
None.

See also
Unpack()
CTDateTime::Second

Syntax
NINT Second() const

Parameters
This method has no parameters.

Description
Retrieves the second component from a CTDateTime object.

Return
Second() returns the unpacked second.

See also
Minute(), Hour(), SecondOfWeek(), Unpack(), Pack()
**Syntax**

```cpp
void SetDateTime(CDATETIME DateTime)
void SetDateTime(const CDateTime& DateTime)
void SetDateTime (CDATE Date, CTIME Time = 0)
void SetDateTime(const CDDate& Date, const CTime& Ti
```

**Parameters**

- *DateTime [in]* The new *DateTime* to set to the *CTDateTime* object.
- *Date [in]* The new date to set to the *CTDateTime* object.
- *Time [in]* The new time to set to the *CTDateTime* object.

**Description**

Sets the *DateTime* to the *CTDateTime* object.

**Return**

None.

**See also**

*GetDateTime()*
CTDateTime::StringToDateTime

Syntax
void StringToDateTime(const CTString& str, CTDATE_TYPE DateType, CTTIME_TYPE TimeType)

Parameters
- **str** [in] The string object to be converted.
- **DateType** [in] The Date type format used to convert to string. The valid Date type formats are shown in "Data Types" (page 163)
- **TimeType** [in] The Time type format used to convert to string. The valid Time type formats are shown in "Data Types" (page 163).

Description
Converts a **CTString** object to a **DateTime** object.

Return
None.

See also
DateTimeToString()
CTDateTime::Unpack

Syntax

```cpp
void Unpack(NINT& year, NINT& month, NINT& day, NINT& hour,
            NINT& minute, NINT& second) const
```

Parameters

- `year` [out] The unpacked year.
- `month` [out] The unpacked month.
- `day` [out] The unpacked day.
- `hour` [out] The unpacked hour.
- `minute` [out] The unpacked minute.
- `second` [out] The unpacked second.

Description

Unpacks a `CTDateTime` type value into hour, minute and second values.

Return

None.

See also

`Pack()`, `Hour()`, `Minute()`, `Second()`, `DayOfWeek()`
CTDateTime::Year

Syntax
NINT Year( ) const

Parameters
This method has no parameters.

Description
Retrieves the year component of a CDateTime object.

Return
Year returns the unpacked year.

See also
Month(), Day(), Unpack(), Pack()
4.9 CTException Class

class CTException

Description

The CTException Class is the base exception handling class for the c-treeDB C++ layer. It is used to throw an error message, when one exception occurs.

Preconditions

This class should be used where an error or unexpected behavior may occur. The error will throw an exception.
CTException Methods

Constructor / Destructor
- **CTException()**: Creates a *CTException* object, and retrieves the error code only.
- **~CTException()**: Destroys a *CTException* Object and resets all the dependent objects.

Error Handling
- **GetErrorCode()**: Returns the last error code
- **GetErrorMsg()**: Returns the last error message
- **GetSourceFile()**: Returns the C++ source file that generated the error
- **GetLineNbr()**: Returns the line number of the C++ source file that generated the error.
CTException::CTException

Syntax
CTException(NINT ErrorCode)
CTException(NINT ErrorCode, pTEXT ErrorMsg)
CTException(NINT ErrorCode, pTEXT SourceFile, NINT LineNbr)
CTException(NINT ErrorCode, pTEXT ErrorMsg, pTEXT SourceFile, NINT LineNbr)

Parameters
- ErrorCode [out] The error code
- ErrorMsg [out] The error message
- SourceFile [out] The source file where the error occurred
- LineNbr [out] The line number where error occurred

Description
This is the constructor for the CTException class.

See also
~CTException()
CTException::~CTException

Syntax
~CTException

Parameters
The destructor has no parameters.

Description
This is the destructor for the CTEexception class.

See also
CTException()
CTException::GetErrorCode

Syntax
NINT GetErrorCode( ) const

Parameters
This method has no parameters.

Description
Returns the last error code.

Return
GetErrorCode() returns the last error code.

See also
GetErrMsg(), GetSourceFile(), GetLineNbr()
CTException::GetErrorMsg

Syntax

```cpp
pTEXT GetErrorMsg( ) const
```

Parameters

This method has no parameters.

Description

Returns a `CTString` object with the last error message. If the error message is not available, `GetErrorMsg()` returns an empty string.

Return

`GetErrorMsg()` returns the last error message.

Example

```cpp
try
{
    pSession->Logon("FAIRCOMS", "ADMIN", "ADMIN");
    printf("\n\nSession Logon successful!\n");
}
catch(CTException& err)
{
    printf("\n\n%d %s", err.GetErrorCode(), err.GetErrorMsg());
}
```

See also

`GetErrorCode()`, `GetSourceFile()`, `GetLineNbr()`
CTException::GetLineNumber

Syntax
NINT GetLineNumber() const

Parameters
This method has no parameters.

Description
Returns the line number of the C++ source file that generated the error, or zero if no line number is available.

Return
GetLineNumber() returns the line where the last error happened.

See also
GetErrorCode(), GetErrorMsg(), GetSourceFile()
CTException::GetSourceFile

Syntax
pTEXT GetSourceFile( ) const

Parameters
This method has no parameters.

Description
Returns the `CString` object with the C++ source file (including path) that generated the last error. If the source file is not available, `GetSourceFile()` returns an empty string.

Return
`GetSourceFile()` returns the source file where the last error happened.

See also
`GetErrorCode()`, `GetErrorMsg()`, `GetLineNbr()`
4.10 **CTField Class**

**class CTField**

**Description**

The *CTField* class deals with the field concept. The field is the basic component of the table. Any table may be composed of as many fields as one needs. It uses *CTBase* as the base class, and implements the constructor/destructor allocating/freeing memory to the field operations.

**See also**

*CTBase, CTSession, CTDatabase, CTTable*

**Preconditions**

Before any task is developed with a *CTField* object, a *CTTable* must have been initialized.

**CTField Operators**

- `=`: Assign a CTField object to another
CTField::operator=

Syntax
CTField& operator=(const CTField& pField)

Parameters
- pField [in] The field object.

Description
Assigns one CTField object to another

Return
Returns a CTField object
CTField Methods

Constructor / Destructor
- **CTField()**: Creates a *CTField* object
- **~CTField()**: Destroys a *CTField* Object and resets all the dependent objects

Field Handling
- **GetLength()**: Retrieves the field length
- **GetName()**: Retrieves the field name
- **GetNullFlag()**: Retrieves the field null flag
- **GetNumber()**: Retrieves the field position index in the table field list
- **GetPrecision()**: Retrieves the field precision
- **GetScale()**: Retrieves the field scale
- **GetType()**: Retrieves the field type
- **IsNumeric()**: Indicates if the field represents a numeric value
- **SetLength()**: Sets the field length
- **SetName()**: Sets the field name
- **SetNullFlag()**: Sets the new null flag value
- **SetPrecision()**: Sets the field precision
- **SetScale()**: Sets the field scale value
- **SetType()**: Sets the field type
CTField::ClearFieldDefaultValue

Clears the default value associated with a field.

Declaration

```cpp
void CTField::ClearFieldDefaultValue();
```

Description

`CTField::ClearFieldDefaultValue()` clears the default value associated with a field. The default date and time types are also reset to their default values of `CTDATE_MDCY` and `CTTIME_HMS` respectively.

Example

```cpp
// clear the default field value
try
{
  CTField hField = hTable.GetField("country");
  hField.ClearFieldDefaultValue();
}
catch (CTException &err)
{
  printf("ClearFieldDefaultValue failed\n");
}
```

Return

void

See Also

`SetFieldDefaultValue()`, `GetFieldDefaultValue()`, `IsFieldDefaultValueSet()`, `ClearAllFieldDefaultValue()`, `SetFieldDefaultDateTimeType()`, `GetFieldDefaultDateType()`, `GetFieldDefaultTimeType()`
CTField::CTField

Syntax
CTField ( )
CTField(const CTField& pField)

Parameters
- pField [in] The field object.

Description
This is the constructor for the CTField object.

See also
- CTFIELD()
CTField::~CTField

Syntax
~CTField

Parameters
This destructor has no parameters.

Description
This is the destructor for the CTFIELD object.

See also
CTField()
CTField::GetFieldDefaultDateType

Retrieves the default value date type.

Declaration

CTDATE_TYPE CTField::GetFieldDefaultDateType();

Description

CTField::GetFieldDefaultDateType() retrieves the default value date type used when converting strings to dates. The default value date type is returned.

Return

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CTDATE_MDCY</td>
<td>Date format is mm/dd/ccyy</td>
</tr>
<tr>
<td>2</td>
<td>CTDATE_MDY</td>
<td>Date format is mm/dd/yy</td>
</tr>
<tr>
<td>3</td>
<td>CTDATE_DMCY</td>
<td>Date format is dd/mm/ccyy</td>
</tr>
<tr>
<td>4</td>
<td>CTDATE_DMY</td>
<td>Date format is dd/mm/yy</td>
</tr>
<tr>
<td>5</td>
<td>CTDATE_CYMD</td>
<td>Date format is ccyymmdd</td>
</tr>
<tr>
<td>6</td>
<td>CTDATE_YMD</td>
<td>Date format is yymmdd</td>
</tr>
</tbody>
</table>

Example

TYPE parameters;

if (FunctionTemplate(parameters))
    printf("\nFunction error = %d",uerr_cod);

See Also

SetFieldDefaultValue(), GetFieldDefaultValue(), IsFieldDefaultValueSet(),
ClearAllFieldDefaultValue(), SetFieldDefaultDateTimeType(), GetFieldDefaultDateType(),
GetFieldDefaultTimeType()
CTField::GetFieldDefaultTimeType

Retrieves the default value time type.

Declaration

```
CTTIME_TYPE CTField::GetFieldDefaultTimeType();
```

Description

**CTField::GetFieldDefaultTimeType()** retrieves the default value time type used when converting string to time. The default value time type is returned.

Return

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CTTIME_HMSP</td>
<td>Time format is hh:mm:ss am</td>
</tr>
<tr>
<td>2</td>
<td>CTTIME_HMP</td>
<td>Time format is hh:mm am</td>
</tr>
<tr>
<td>3</td>
<td>CTTIME_HMS</td>
<td>Time format is hh:mm:ss (24 hour)</td>
</tr>
<tr>
<td>4</td>
<td>CTTIME_HM</td>
<td>Time format is hh:mm (24 hour)</td>
</tr>
<tr>
<td>5</td>
<td>CTTIME_MIL</td>
<td>Time format is hhmm (military)</td>
</tr>
</tbody>
</table>

Example

```c
// check the default date type
try
{
    CTField hField = hTable.GetField("country");
    if (hField.GetFieldDefaultTimeType() == CTTIME_HMS)
        printf("Field default time type is OK\n");
}
catch (CTException &err)
{
    printf("GetFieldDefaultTimeType() failed\n");
}
```

See Also

`SetFieldDefaultValue()`, `GetFieldDefaultValue()`, `IsFieldDefaultValueSet()`, `ClearAllFieldDefaultValue()`,
**CTField::GetFieldDefaultValue**

Retrieves the current field default value.

**Declaration**

```cpp
VRLEN GetFieldDefaultValue(pTEXT& value);
VRLEN GetFieldDefaultValue(CTString& value);
```

**Description**

`CTField::GetFieldDefaultValue()` retrieves the current field default value. If no default value is set `CTField::GetFieldDefaultValue()` returns zero. You can use `CTField::IsFieldDefaultValueSet()` method to check if a field default value is set or not. Parameter `value` receives the content of the field default value.

**Return**

Returns the number of bytes copied to value parameter. If no default value is available, the returned value is zero.

**Example**

```cpp
// check if default field value is 'USA'
try
{
    CTString value;
    CTField hField = hTable.GetField("country");

    hField = ctdbGetField(hTable, 5);
    if (hField.GetFieldDefaultValue(value) > 0)
    {
        if (value == "USA")
            printf("Default value is USA\n");
        else
            printf("Default value is not USA\n");
    }
    else
        printf("No default value set\n");
}
catch (CTException &err)
{
    printf("GetFieldDefaultValue failed\n");
}
```

**See Also**

`SetFieldDefaultValue()`, `GetFieldDefaultValue()`, `IsFieldDefaultValueSet()`, `ClearAllFieldDefaultValue()`, `SetFieldDefaultDateTimeType()`, `GetFieldDefaultDateType()`, `GetFieldDefaultTimeType()`
CTField::GetLength

Syntax
VRLEN GetLength() const

Parameters
This method has no parameters.

Description
Retrieves the field length.

Return
GetLength() retrieves the field length.

See also
GetName(), GetType(), SetLength()
CTField::GetName

Syntax

CTString GetName( ) const

Parameters

This method has no parameters.

Description

Retrieves the field name.

Return

GetName() retrieves the field name.

See also

GetName(), GetType(), GetLength(), SetName()
**CTField::GetNullFlag**

**Syntax**

```cpp
CTBOOL GetNullFlag( ) const
```

**Parameters**

This method has no parameters.

**Description**

Retrieves the field null flag.

The c-treeDB null flag controls the NOT NULL property of a column. Setting this column has no effect on individual record null values: it is NOT enforced at the c-treeDB layer. This attribute only applies to the c-treeACE SQL layer for constraint on values. It is useful to set this flag in c-tree data files before SQL import such that the property is maintained.

**Return**

- `GetNullFlag()` returns the field null flag setting.

**See also**

- `SetNullFlag()`
CTField::GetNumber

Syntax
NINT GetNumber();

Parameters
This method has no parameters.

Description
Retrieves the field position index in the table field list.

Return
GetNumber() returns the field number.
**CTField::GetPrecision**

**Syntax**

NINT GetPrecision( ) const

**Parameters**

This method has no parameters.

**Description**

Retrieves the field precision (maximum number of digits).

**Return**

GetPrecision() returns the field precision value.

**See also**

SetPrecision()
CTField::GetScale

Syntax
NINT GetScale( ) const

Parameters
This method has no parameters.

Description
Retrieves the field scale (the number of digits to the right of the decimal point).

Return
GetScale() returns the field scale value.

See also
SetScale()
CTField::GetStatus

 Retrieves the changed status of a field object.

 Declaration

 ULONG CTField::GetStatus() const;

 Description

 CTField::GetStatus() retrieves the changed status of a field object. The status of a field object is a bit map describing one or more changes that have occurred with the field object.

 Return

 CTField::GetStatus returns a bitmap of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>CTDBFIELD_OLD</td>
<td>Original field as read from table</td>
</tr>
<tr>
<td>0x01</td>
<td>CTDBFIELD_NEW</td>
<td>Field added or inserted</td>
</tr>
<tr>
<td>0x02</td>
<td>CTDBFIELD_DEL</td>
<td>Original field deleted</td>
</tr>
<tr>
<td>0x04</td>
<td>CTDBFIELD_MOVED</td>
<td>Original field moved</td>
</tr>
<tr>
<td>0x10</td>
<td>CTDBFIELD_NAME</td>
<td>Field name changed</td>
</tr>
<tr>
<td>0x20</td>
<td>CTDBFIELD_TYPE</td>
<td>Field type changed</td>
</tr>
<tr>
<td>0x40</td>
<td>CTDBFIELD_LEN</td>
<td>Field length changed</td>
</tr>
<tr>
<td>0x80</td>
<td>CTDBFIELD_RESOURCE</td>
<td>Field resource changed</td>
</tr>
</tbody>
</table>

 Example

 // if field is new delete it
 for (i = 0; i < (NINT) hTable.GetFieldCount(); i++)
 {
   CTField hField = hTable.GetField(i);
   if (hField.GetStatus() & CTDBFIELD_NEW)
     hTable.DelField(i);
 }

 See Also

 CTIndex::GetStatus(), CTSegment::GetStatus()
CTField::GetType

Syntax
cterdb::CTDBTYPE GetType( ) const

Parameters
This method has no parameters.

Description
Retrieves the field type.

Return
GetType() retrieves the field type.

See also
GetName(), GetLength(), SetType()
CTField::IsFieldDefaultValueSet

Checks if a field default value has been set or not.

Declaration

CTBOOL CTField::IsFieldDefaultValueSet()

Description

CTField::IsFieldDefaultValueSet() checks if a field default value has been set or not. This method returns YES if a field default value was set, otherwise it returns NO.

Return

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO</td>
<td>Field default value is not set.</td>
</tr>
<tr>
<td>1</td>
<td>YES</td>
<td>Field default value is set.</td>
</tr>
</tbody>
</table>

Example

// check if default field value is set
CTField hField = hTable.GetField("country");
if (hField.IsFieldDefaultValueSet())
    printf("Default field value is set\n");
else
    printf("No default field value\n");

See Also

SetFieldDefaultValue(), GetFieldDefaultValue(), ClearFieldDefaultValue(),
ClearAllFieldDefaultValue(), SetFieldDefaultDateTimeType(), GetFieldDefaultDateType(),
GetFieldDefaultTimeType()
CTField::IsNumeric

Syntax

CTBOOL IsNumeric( ) const

Parameters

This method has no parameters.

Description

Indicates if the field represents a numeric value.

Return

IsNumeric() returns YES if the field is numeric.

See also

SetNullFlag()
CTField::SetFieldDefaultDateTimeType

Sets the default field value date and time type.

Declaration

void CTField::SetFieldDefaultDateTimeType(CTDATE_TYPE dateType, CTTIME_TYPE timeType);

Description

CTField::SetFieldDefaultDateTimeType() sets the default field value date and time type to be used when converting CT_DATE, CT_TIME and CT_TIMES string values. By default the date type is CTDATE_MDCY and the time type is CTTIME_HMS. Use this method to modify the default values.

The CTField::SetFieldDefaultDateTimeType() method modifies both the date and time types. If you wish to change only the default date time, but keep the current time type, use the following example:

hField.SetFieldDefaultDateTimeType(CTDATE_YMD, hField.GetFieldDefaultTimeType());

You can use the same approach to change only the time type, keeping the current date type:

hField.SetFieldDefaultDateTimeType(ctdbGetFieldDefaultDateType(), CTIME_HMP);

dateType is the date type to be used for converting values between dates and strings. The possible values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CTDATE_MDCY</td>
<td>Date format is mm/dd/ccyy</td>
</tr>
<tr>
<td>2</td>
<td>CTDATE_MDY</td>
<td>Date format is mm/dd/yy</td>
</tr>
<tr>
<td>3</td>
<td>CTDATE_DMCY</td>
<td>Date format is dd/mm/ccyy</td>
</tr>
<tr>
<td>4</td>
<td>CTDATE_DMY</td>
<td>Date format is dd/mm/yy</td>
</tr>
<tr>
<td>5</td>
<td>CTDATE_CYMD</td>
<td>Date format is ccyymmdd</td>
</tr>
<tr>
<td>6</td>
<td>CTDATE_YMD</td>
<td>Date format is yymmdd</td>
</tr>
</tbody>
</table>

timeType is the time type to be used for converting values between time and strings. The possible values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CTTIME_HMSP</td>
<td>Time format is hh:mm:ss am</td>
</tr>
<tr>
<td>2</td>
<td>CTTIME_HMP</td>
<td>Time format is hh:mm am</td>
</tr>
<tr>
<td>3</td>
<td>CTTIME_HMS</td>
<td>Time format is hh:mm:ss (24 hour)</td>
</tr>
<tr>
<td>4</td>
<td>CTTIME_HM</td>
<td>Time format is hh:mm (24 hour)</td>
</tr>
<tr>
<td>5</td>
<td>CTTIME_MIL</td>
<td>Time format is hhmm (military)</td>
</tr>
</tbody>
</table>

Example
// set the field default date and time types
try{
CTField hField = hTable.GetField("country");
hField.SetFieldDefaultDateTimeType();
}
catch (CTException &err)
{
printf("SetFieldDefaultDateTimeType failed\n");
}

See Also
SetFieldDefaultValue(), GetFieldDefaultValue(), IsFieldDefaultValueSet(),
ClearFieldDefaultValue(), ClearAllFieldDefaultValue(), GetFieldDefaultDateType(),
GetFieldDefaultTimeType()
**CTField::SetFieldDefaultValue**

Sets the field default value.

**Declaration**

```cpp
void CTField::SetFieldDefaultValue(const CTString& value);
void CTField::SetFieldDefaultValue(pTEXT value, VRLEN length);
```

**Description**

- **value** is a string with the default value. No checks are made to make sure the default value matches the correct field type. The caller is responsible for passing the appropriate value.
- **length** is the length of the string. You must pass a proper length of the string.

**CTField::SetFieldDefaultValue()** sets the field default value. The default value of a field is used during an alter table operation when a full table rebuild is performed. During a full alter table rebuild, and after the old record buffer data is moved to the new record buffer, the new record buffer is scanned and if a NULL field is found and that NULL field has a default value, the default value is copied to the field buffer.

The field default value is kept as a string representation of the data. It is recommended that numeric data should be converted to string using one of the rich set of c-treeDB data conversion functions.

Date values should be converted to string using the default date type value. The default date type value can be retrieved by calling **CTField::GetFieldDefaultDateType()** method. By default, the date type is **CTDATE_MDCY**.

Time values should be converted to string using the default time type value. The default time type value can be retrieved by calling **CTField::GetFieldDefaultTimeType()** function. By default, the time type is **CTTIME_HMS**.

Time stamp values should be converted to string using the default date type and time type values as described above.

**Return**

void

**Example**

```cpp
// set the default value of country field */
try
{
    CTField hField = hTable.GetField("country");
    hField.SetFieldDefaultValue("USA");
}
catch (CTException &err)
{
    printf("SetFieldDefaultValue failed\n");
}
```
See Also

SetFieldDefaultValue(), GetFieldDefaultValue(), ClearFieldDefaultValue(),
IsFieldDefaultValueSet(), ClearAllFieldDefaultValue(), SetFieldDefaultDateTimeType(),
GetFieldDefaultDateTimeType(), GetFieldDefaultTimeType()
CTField::SetLength

Syntax
void SetLength(const VRLEN Length)

Parameters
- **Length** [in] The field length to be set to the field.

Description
Sets the field length

Return
None.

See also
GetLength(), SetName(), SetType()
**CTField::SetName**

**Syntax**

```cpp
void SetName(const CTString& FieldName)
```

**Parameters**

- *FieldName* [in] The field name to be set to the field.

**Description**

Sets the field name.

**Return**

None.

**See also**

`GetName()`, `SetType()`, `SetLength()`
CTField::SetNullFlag

Syntax

void SetNullFlag(CTBOOL flag)

Parameters


Description

Sets the new null flag value. The null flag indicates if the field can contain a null value.

The c-treeDB null flag controls the NOT NULL property of a column. Setting this column has no effect on individual record null values: it is NOT enforced at the c-treeDB layer. This attribute only applies to the c-treeACE SQL layer for constraint on values. It is useful to set this flag in c-tree data files before SQL import such that the property is maintained.

Return

None.

See also

GetNullFlag()
CTField::SetPrecision

Syntax
void SetPrecision(NINT precision)

Parameters

Description
Sets the field precision (maximum number of digits).

Return
None.

See also
GetPrecision()
CTField::SetScale

Syntax
void SetScale(NINT scale)

Parameters
- scale [in] The new field scale value.

Description
Sets the field scale (the number of digits to the right of the decimal point).

Return
None.

See also
GetScale()
CTField::SetType

Syntax
void SetType(const CTDBTYPE Type, const VRLEN Length=0)

Parameters
- **Type** [in] The field type to be set to the field. Valid type values are: CTBIGINT, CTBLOB, CTBOOL, CTCURRENCY, CDATE, CDATETIME, CTFLOAT, CTMONEY, CNUMBER, CTSIGNED, CSTRING, CTIME and CTUNSIGNED.
- **Length** [in] The field length to be set to the field.

Description
Sets the field type.

Return
None.

See also
GetType(), SetName(), SetLength()
4.11 CTIndex Class

class CTIndex

Description
The CTIndex class deals with the index concept. The index is part of the table, and represents one field, or parts of one field, or parts of several fields. One single index may be composed of several segments.

See Also
CTBase, CTTable, CTRecord, CTField, CTSegment

Preconditions
Before any task is developed with a CTIndex object, a CTTable must have been initialized.

CTIndex Operators
- = : Assign a CTIndex object to another
CTIndex::operator=

Syntax
CTIndex& operator=(const CTIndex& pField)

Parameters

Description
Assigns one CTIndex object to another

Return
Returns a CTIndex object
CTIndex Methods

Constructor / Destructor
- **CTIndex()**: Creates a *CTIndex* object
- **~CTIndex()**: Destroys a *CTIndex* Object and resets all the dependent objects

Key Handling
- **GetKeyLength()**: Retrieves the key length
- **GetKeyType()**: Retrieves the key type
- **GetEmptyChar()**: Retrieves the index empty char
- **GetDuplicateFlag()**: Retrieves the allow duplicated flag for this index
- **GetNullFlag()**: Retrieve the null flag
- **GetTemporaryFlag()**: Retrieves the temporary flag
- **SetEmptyChar()**: Sets the empty char value.
- **SetDuplicateFlag()**: Sets the allow duplicate flag
- **SetNullFlag()**: Sets the null flag.
- **SetKeyType()**: Set the index key type.
- **SetTemporaryFlag()**: Sets the temporary flag.

Segment Handling
- **GetSegmentCount()**: Retrieves the index segments count
- **AddSegment()**: Adds a new segment to the index
- **InsertSegment()**: Inserts a segment to the index
- **DelSegment()**: Deletes a segment from the index
- **GetSegment()**: Retrieves an index segment

Index Handling
- **GetIdxno()**: Retrieves a table index number from an index object.
- **GetNumber()**: Retrieves the index position in table indices list.
- **GetName()**: Retrieves the index name.
- **GetUID()**: Retrieves the index uid.
- **GetIndexFileName()**: Retrieves the index file name for the associated index.
- **SetIndexFileName()**: Sets the index file name for the associated index.
CTIndex::CTIndex

Syntax
CTIndex ( )
CTIndex(const CTIndex& pIndex)

Parameters
- pIndex [in] The index object.

Description
This is the constructor for the CTIndex object.

See also
~CTIndex()
CTIndex::~CTIndex

Syntax
~CTIndex

Parameters
This destructor has no parameters.

Description
This is the destructor for the CTIndex object.

See also
CTIndex()
CTIndex::AddSegment

Syntax
CTSegment AddSegment(const CTFIELD & pField, CTSEG_MODE SegMode)
CTSegment AddSegment(NINT offset, NINT length, CTSEG_MODE SegMode)

Parameters
- **pField** [in] The segment object to be inserted.
- **SegMode** [in] The field segment mode. The valid values for the segment mode are shown in "c-treeDB definitions" (page 227).
- **offset** [in] The segment offset to be inserted.
- **length** [in] The segment length to be inserted.

Description
Adds a new segment to the index at the end of the index.

Return
Returns a segment object

See also
InsertSegment(), DelSegment(), GetSegmentCount()
### CTIndex::DelSegment

**Syntax**

```cpp
void DelSegment(NINT SegmentNumber)
```

**Parameters**

- **SegmentNumber** [in] The segment number to be deleted.

**Description**

Deletes a segment from the index.

**Return**

None.

**See also**

`AddSegment()`, `InsertSegment()`
CTIndex::GetDuplicateFlag

Syntax
CTBOOL GetDuplicateFlag( ) const

Parameters
This method has no parameters.

Description
Retrieves the allow duplicated flag for this index

Return
GetDuplicateFlag() returns YES if the index allows duplicated keys, and NO otherwise.

See also
SetDuplicateFlag()
CTIndex::GetEmptyChar

Syntax
NINT GetEmptyChar() const

Parameters
This method has no parameters.

Description
Retrieves the index empty char

Return
GetEmptyChar() returns the index empty char. The empty char property is expressed as the decimal equivalent of the ASCII table. See SetEmptyChar() for more information.

See also
SetEmptyChar(), GetNullFlag()
CTIndex::GetIndexFileName

Returns the name of the index file name for the given index name or number.

Declaration

CTString CTTable::GetIndexFileName(NINT IndexNumber)
CTString CTTable::GetIndexFileName(const CTString& IndexName)

Description

Retrieves the file pathname for the given index name or number. GetIndexFileName() accepts as parameters either the index number or the index name.

Return

The index file pathname is returned. If the index has no file name, for example, the index is a member of an index file, GetIndexFileName() returns an empty string.

Example

void SetFirstIndexName(CTTable& htable, const CTString idxFileName)
{
    try
    {
        if (htable.GetIndexFileName(0) != idxFileName)
        {
            hTable.SetIndexFileName(0, idxFileName);
            hTable.Alter(CTDB_ALTER_NORMAL);
        }
    }
    catch (CTException &err)
    {
        hSession.Abort();
        printf("Error: %d - %s\n", err.GetErrorMsg(), err.GetErrorCode());
    }
}

See Also

CTTable:GetIndexFilename(), CTTable::SetIndexFilename, CTIndex::SetIndexFilename
CTIndex::GetIndexKSeg

Retrieves the current index wide extended key segment definition.

Declaration

void CTIndex::GetIndexKSeg(ctKSEGDEF pKSeg);

Description

CTIndex::GetIndexKSeg() retrieves the current index wide extended key segment definition. 

pKSeg is a pointer to an extended key segment definition structure which will receive the
definition.

Return

void

Example

ctKSEGDEF kseg;
hIndex.GetIndexKSeg(&kseg);

See Also

CTIndex::SetIndexKSeg(), CTSegment::GetSegmentKSeg(),
CTSegment::SetKSegDefaults(),
CTSegment::SetSegmentKSeg(), CTTable::GetTableKSeg(), CTTable::SetTableKSeg()
CTIndex::GetIdxno

Retrieves a table index file number from an index object.

**Declaration**

NINT CTIndex::GetIdxno()

**Description**

Retrieves a table index file number from an index object.

**Return**

GetIdxno() returns a table index file number.

**Example**

```c
// retrieve the first key of first index
TEXT keyval[256];
CTIndex hIndex = hTable.GetIndex(0);
if (FirstKey(hIndex.GetIdxno(), keyval)
    printf("FirstKey failed\n");
```

**See Also**

CTBase::SwitchInstance(), CTRecord::SwitchContext(), CTTable::GetDatno(), CTTable::GetIdxno()
CTIndex::GetKeyLength

Syntax
NINT GetKeyLength( ) const

Parameters
This method has no parameters.

Description
Retrieves the key length.

Return
GetKeyLength() returns the key length.

See also
GetKeyType()
CTIndex::GetKeyType

Syntax
CTDBKEY GetType() const

Parameters
This method has no parameters.

Description
Retrieves the key type.

Return
GetType() returns the key type.

See also
getKeyLength()
CTIndex::GetName

Syntax

CTString GetName( )

Parameters

This method has no parameters.

Description

Retrieves the index name.

Return

GetName() returns the index name.

See also

GetNumber(), GetSegment()
CTIndex::GetNullFlag

Syntax
CTBOOL GetNullFlag( ) const

Parameters
This method has no parameters.

Description
Retrieves the null flag.

Return
GetNullFlag() returns YES if the null flag is set, and NO otherwise

See also
SetNullFlag(), GetEmptyChar()
CTIndex::GetNumber

Syntax
NINT GetNumber()

Parameters
This method has no parameters.

Description
Retrieves the index position in table indices list.

Return
GetNumber() returns the index position.

See also
GetName()
CTIndex::GetSegment

Syntax

CTSegment GetSegment(NINT SegmentNumber)

Parameters

- **SegmentNumber** [in] The segment number to be retrieved.

Description

Retrieves an index segment.

Return

GetSegment() returns a CTSegment object.

See also

GetSegmentCount(), GetName(), GetNumber()
CTIndex::GetSegmentCount

Syntax
NINT GetSegmentCount() const

Parameters
This method has no parameters.

Description
Retrieves the index segments count.

Return
GetSegmentCount() returns the number of segments in the index.

See also
GetSegment()
CTIndex::GetStatus

Retrieves the status of the index object.

Declaration

ULONG CTIndex::GetStatus() const;

Description

CTIndex::GetStatus() retrieves the status of a index object. The status of the index object is a bit map describing one or more changes that have occurred to the index object.

Return

CTIndex::GetStatus() returns a bitmap of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>CTDBINDEX_OLD</td>
<td>Original value (no changes)</td>
</tr>
<tr>
<td>0x01</td>
<td>CTDBINDEX_NEW</td>
<td>Index added</td>
</tr>
<tr>
<td>0x02</td>
<td>CTDBINDEX_DEL</td>
<td>Original Index deleted</td>
</tr>
<tr>
<td>0x04</td>
<td>CTDBINDEX_KEYTYPE</td>
<td>Index key type changed</td>
</tr>
<tr>
<td>0x10</td>
<td>CTDBINDEX_EMPCHAR</td>
<td>Index empty char changed</td>
</tr>
<tr>
<td>0x20</td>
<td>CTDBINDEX_DUPFLAG</td>
<td>Index duplicate flag changed</td>
</tr>
<tr>
<td>0x40</td>
<td>CTDBINDEX_NULLFLAG</td>
<td>Index null flag changed</td>
</tr>
<tr>
<td>0x80</td>
<td>CTDBINDEX_AIDXNAM</td>
<td>Index file name changed</td>
</tr>
</tbody>
</table>

Example

```cpp
// if the index has been changed, call alter table
CTIndex hIndex = hTable.GetIndex(0);

if (hIndex.GetStatus() != CTDBINDEX_OLD)
    if (hTable.Alter(CTDB_ALTER_NORMAL);
```

See Also

CTField::GetStatus(), CTSegment::GetStatus()
CTIndex::GetTemporaryFlag

Syntax

CTBOOL GetTemporaryFlag( ) const

Parameters

This method has no parameters.

Description

Retrieves the temporary flag.

Return

GetTemporaryFlag() returns YES if the index is temporary, NO otherwise

See also

SetTemporaryFlag()
**CTIndex::GetUID**

Syntax

```cpp
ULONG GetUID( )
```

Parameters

This method has no parameters.

Description

Retrieves the index UID.

Return

GetUID() returns the index UID.

See also

GetName()
CTIndex::InsertSegment

Syntax

CTSegment InsertSegment(NINT BeforeSegment, const CTField& Field,
                          CTSEG_MODE SegMode)
CTSegment InsertSegment(NINT BeforeSegment, NINT offset, NINT length,
                          CTSEG_MODE SegMode)

Parameters

- **BeforeSegment** [in] Insert the new segment before this segment.
- **Field** [in] The segment object to be inserted.
- **SegMode** [in] The field segment mode. The valid values for the segment mode are shown in "c-treeDB definitions" (page 227).
- **offset** [in] The segment offset to be inserted.
- **length** [in] The segment length to be inserted.

Description

Inserts a new segment to the index, in a specified position.

Return

Returns a segment object

See also

AddSegment(), DelSegment(), GetSegmentCount()
**CTIndex::MoveSegment**

Moves a key segment.

**Declaration**

```cpp
void CTIndex::MoveSegment(NINT segmentNumber, NINT newIndex);
```

**Description**

*CTIndex::MoveSegment()* moves a key segment to a location indicated by *newIndex*.

- *segmentNumber* is a relative number for a segment in an index definition.
- *newIndex* indicates the relative position were the key segment should be moved to.

**Return**

None.

**Example**

```cpp
// move the last segment to first
CTIndex hIndex = hTable.GetIndex(0);
NINT count = hIndex.GetSegmentCount();

if (count > 0)
    hIndex.MoveSegment((count - 1), 0);
```

**See Also**

*CTTable::MoveSegment()* , *CTSegment::MoveSegment()*
CTIndex::SetDuplicateFlag

Syntax
void SetDuplicateFlag(CTBOOL DupFlag)

Parameters
- **DupFlag** [in] The allow duplicates flag

Description
Sets the allow duplicates flag. If set to YES, this index allows duplicated values.

Return
None.

See also
GetDuplicateFlag()
CTIndex::SetEmptyChar

Syntax

```cpp
void SetEmptyChar(NINT EmptyChar)
```

Parameters

- `EmptyChar [in]` The empty char value. The empty char property is expressed as the decimal equivalent of the ASCII table. For instance, an ASCII space is specified as 32, and a NULL byte is specified as 0.

Description

Sets the empty char property.

Return

None.

See also

`GetEmptyChar()`, `SetNullFlag()`
CTIndex::SetIndexFileName

Sets the index file name of the given index name or number.

Declaration
void CTTable::SetIndexFilename(const CTString& IndexName, const CTString* path, const CTString* filename)
void CTTable::SetIndexFilename(NINT IndexNumber, const CTString* path, const CTString* filename)

Description
Sets the index file path and name of the given index name or number. By default, when a table is created, all indices are created as members of one index file with the table name and path, and with the extension .idx. SetIndexFileName() allows the specification of a file name and path for any one of the table’s indices. If an index file name is not specified, the index will be created as a member of the previous index file. If no index file name and path is specified, the default is applied. If all indices of a table specify an index file name and path, each index will be placed in a separate index file.

- **path** specifies the directory were the index file is located. If **path** is NULL the index is located in the same directory of its table.
- **Indexname** specifies the name of index file. If **name** is NULL the index file will use the name of its table.
- **filename** is the name to assign to the index file.

Return
void

Example

void CreateTable(CTDatabase& hDatabase, const CTString& tabName, const CTString& idxName)
{
    CTTable hTable(hDatabase);
    try
    {
        hTable.AddField("name", CT_FSTRING, 20);
        hTable.AddField("surname", CT_FSTRING, 20);
        hTable.AddField("code", CT_INT4, 4);
        hTable.AddIndex("index1", CTINDEX_FIXED, NO, NO);
        hTable.SetIndexFilename("index1", NULL, IdxName);
        hTable.AddSegment("index1", "surname", CTSEG_SCHSEG);
        hTable.AddSegment("index1", "code", CTSEG_SCHSEG);
        hTable.Create(tabName, CTCREATE_NORMAL);
    }
    catch (CTException &err)
    {
        printf("Error: %d - %s\n", err.GetErrorMsg(), err.GetErrorCode());
    }
}
See Also
CTTable:GetIndexFilename(), CTTable::SetIndexFilename, CTIndex::SetIndexFilename
**CTIndex::SetIndexKSeg**

Establishes an index-wide extended key segment definition.

**Declaration**

```cpp
void CTIndex::SetIndexKSeg(pctKSEGDEF pKSeg);
```

**Description**

`CTIndex::SetIndexKSeg()` establishes an index wide extended key segment definition. `pKSeg` is a pointer to an extended key segment definition structure with the extended key definition.

**Return**

`void`

**Example**

```cpp
cTKSEGDEF kseg;

kseg.kseg_ssiz = ctKSEG_SSIZ_COMPUTED;
kseg.kseg_type = ctKSEG_TYPE_UNICODE;
kseg.kseg_styp = ctKSEG_STYP_UTF16;
kseg.kseg_comp = ctKSEG_COMPU_S_DEFAULT | ctKSEG_COMPU_N_NONE;
kseg.kseg_desc = "en_US"

hIndex.SetIndexKSeg(&kseg);
```

**See Also**

- `CTIndex::GetIndexKSeg()`, `CTSegment::GetSegmentKSeg()`,
- `CTSegment::SetKSegDefaults()`,
- `CTSegment::SetSegmentKSeg()`, `CTTable::GetTableKSeg()`, `CTTable::SetTableKSeg()`
**CTIndex::SetKeyType**

Set the index key type.

**Declaration**

```cpp
void CTIndex::SetKeyType(CTDBKEY keytype)
```

**Description**

`CTIndex::SetKeyType()` sets the key type for this index. Use `CTIndex::Add()` to add an index to a table. Use the `CTIndex::GetKeyType()` to retrieve the Index key type.

- **KeyType** [in] the key type

Valid `keytype` values are:

- `CTINDEX_FIXED`: Fixed-length key
- `CTINDEX_LEADING`: Leading-character compression
- `CTINDEX_PADDING`: Padding compression
- `CTINDEX_LEADPAD`: Leading and padding compression

**Note**: Key compression imposes a significant performance impact, especially when deleting records. Use this feature only when absolutely necessary to keep index space requirements to a minimum.

**Return**

`SetKeyType()` does not return a value but will throw an exception if the index key type cannot be set.

See "c-treeDB Errors and Return Values" (page 881) for a complete listing of valid c-treeDB error codes and return values.

**See Also**

`CTIndex::GetKeyType()`, `CTIndex::Add()`
CTIndex::SetNullFlag

Syntax

```cpp
void SetNullFlag(CTBOOL NullFlag)
```

Parameters


Description

Sets the null flag. If set to YES, the system checks for NULL or missing key values after the index has been concatenated.

Return

None.

See also

GetNullFlag(). SetEmptyChar()
CTIndex::SetTemporaryFlag

Syntax
void SetTemporaryFlag(CTBOOL TempFlag)

Parameters

Description
Sets the temporary flag. If set to YES, this index is a temporary index.

Return
None.

See also
GetTemporaryFlag()
4.12 CTMoney Class

class CTMoney

Description
The CTMoney class represents Money objects.

See Also
CTString, CTCurrency

Preconditions
This is one of the basic objects from the c-tree database layer.

CTMoney Operators
- operator =, +=, -=, *=, /=, abs, +, -, *, / : Assignment and arithmetic operators
- operator <, <=, >, >=, ==, != : Comparison operators
CTMoney::operator =, +=, -=, *=, /=, abs, +, -, *, /

Syntax
CTMoney& operator=(CTFLOAT value)
CTMoney& operator=(CTMONEY value)
CTMoney& operator=( const CTString& str)
CTMoney& operator=(const CTMoney& money)
CTMoney& operator +(const CTMoney& money)
CTMoney& operator -(const CTMoney& money)
CTMoney& operator *(const CTMoney& money)
CTMoney& operator /(const CTMoney& money)
CTMoney& operator abs(const CTMoney& money)
CTMoney& operator +(const CTMoney& left, const CTMoney& right)
CTMoney& operator -(const CTMoney& left, const CTMoney& right)
CTMoney& operator *(const CTMoney& left, const CTMoney& right)
CTMoney& operator /(const CTMoney& left, const CTMoney& right)

Parameters
- value [in] The double value to be converted and assigned to the new CTMoney object
- str [in] The CTString object to be assigned to the new CTMoney object
- money [in] The CTMoney object to be assigned to the new CTMoney object
- left [in] The left CTMoney object to be concatenated to form the new CTMoney object
- right [in] The right CTMoney object to be concatenated to form the new CTMoney object

Description
Assigns or concatenates values to form a CTMoney object

Return
The new CTMoney object
CTMoney::operator <, <=, >, >=, ==, !=

Syntax
CTBOOL operator < (const CTMoney& left, const CTMoney& right)
CTBOOL operator <= (const CTMoney& left, const CTMoney& right)
CTBOOL operator > (const CTMoney& left, const CTMoney& right)
CTBOOL operator >= (const CTMoney& left, const CTMoney& right)
CTBOOL operator == (const CTMoney& left, const CTMoney& right)
CTBOOL operator != (const CTMoney& left, const CTMoney& right)

Parameters
- left [in] The left CTMoney object to be compared
- right [in] The right CTMoney object to be compared

Description
These overloaded operators make comparisons between two CTMoney objects.

Return
The operators return YES or NO, depending on the result of the comparison.
CTMoney Methods

Constructor / Destructor

- `CTMoney()`: Creates a `CTMoney` object
- `~CTMoney()`: Destroys a `CTMoney` Object and resets all the dependent objects

Money Handling

- `SetMoney()`: Initializes the `CTMoney` object.
- `AsFloat()`: Converts a `CTMoney` object to float
- `AsLong()`: Converts a `CTMoney` object to long
- `AsMoney()`: Converts a `CTMoney` object to `CTMoney`
- `AsString()`: Converts a `CTMoney` object to a `CTString` object
- `StringToMoney()`: Converts a `CTString` object to a `CTMoney` object
CTMoney::CTMoney

Syntax

CTMoney ( )
CTMoney (CTFLOAT value)
CTMoney (CTMONEY money)
CTMoney (const CTString& str)
CTMoney (CTMoney& money)

Parameters

- **Value** [in] The double value to be converted and assigned to the new CTMoney object
- **money** [in] The CTMoney object or Money to be assigned to the new CTMoney object
- **str** [in] The CTString object to be assigned to the new CTMoney object

Description

This is the constructor for the CTMoney object.

See also

~CTMoney()
CTMoney::~CTMoney

Syntax
~CTMoney

Parameters
This destructor has no parameters.

Description
This is the destructor for the CTMoney object.

See also
CTMoney
CTMoney::AsFloat

Syntax

CTFLOAT AsFloat() const

Parameters

This method has no parameters.

Description

Converts a CTMoney object to float.

Return

AsFloat() returns the converted value in CTFloat format.

See also

AsLong()
CTMoney::AsLong

Syntax
LONG AsLong( ) const

Parameters
This method has no parameters.

Description
Converts a CTMoney object to LONG.

Return
AsLong() returns the converted value in LONG format.

See also
AsFloat()
CTMoney::AsMoney

Syntax
CTMONEY AsMoney() const

Parameters
This method has no parameters.

Description
Converts a CTMONEY object to a CTMoney object.

Return
AsMoney() returns a CTMoney object.

See also
AsString()
CTMoney::AsString

Syntax
CTStringAsString( ) const

Parameters
This method has no parameters.

Description
Converts a CTMONEY object to a CTString object.

Return
AsString() returns a CTString object with the Money value.

See also
StringToMoney(). AsMoney()
CTMoney::SetInt

Declaration
CTMoney::SetInt(int value);

Description
Enable the assignment of an integer value to a CTMoney object.

Return
None.
CTMoney::SetMoney

Syntax
void SetMoney(CTFLOAT value)
void SetMoney(CTMONEY money)
void SetMoney(const CTString& str)
void SetMoney(const CTMoney& money)

Parameters
- **value** [in] The double value to be converted and assigned to the CTMoney object
- **money** [in] The CTMoney object or Money to be assigned to the CTMoney object
- **str** [in] The CTString object to be assigned to the CTMoney object

Description
Initializes the CTMoney object.

Return
None.
CTMoney::StringToMoney

Syntax
void StringToMoney(const CTString& str)

Parameters
- str [in] The string object to be converted.

Description
StringToMoney() converts a CTString object to a CTMoney object.

Return
None.

See also
AsString()
4.13 CTNumber Class

class CTNumber

Description
The CTNumber class represents number objects.

Preconditions
This is one of the basic objects from the c-tree database layer.

CTNumber Operators
- operator =, +=, -=, *=, /=, abs, +, -, *, : Assignment and arithmetic operators
- operator <, <=, >, >=, ==, != : Comparison operators
**CTNumber::operator =, +, -,(, /, abs**

**Syntax**

CTNumber& operator=(LONG value)
CTNumber& operator=(CTFLOAT value)
CTNumber& operator=(pCTNUMBER value)
CTNumber& operator=(const CTMoney& value)
CTNumber& operator=(const CTString& value)
CTNumber& operator=(const CTBigint& value)
CTNumber& operator=(const CTCurrency& value)
CTNumber& operator=(const CTNumber& value)
CTNumber& operator+(const CTNumber& left, const CTNumber & right)
CTNumber& operator-(const CTNumber& left, const CTNumber & right)
CTNumber& operator*(const CTNumber& left, const CTNumber & right)
CTNumber& operator/(const CTNumber& left, const CTNumber & right)
CTNumber abs(const CTNumber& value)
CTNumber operator+(const CTNumber& left, const CTNumber& right)
CTNumber operator-(const CTNumber& left, const CTNumber& right)
CTNumber operator*(const CTNumber& left, const CTNumber& right)
CTNumber operator/(const CTNumber& left, const CTNumber& right)

**Parameters**

- **value** [in] long, double, **CTNumber, CTMoney, CTBigint, CTCurrency, or CTString** value or object to be assigned to or updated with the new **CTNumber** object.
- **left** [in] The left **CTNumber** object to be concatenated to form the new **CTNumber** object
- **right** [in] The right **CTNumber** object to be concatenated to form the new **CTNumber** object

**Description**

Assigns or concatenates values to form a **CTNumber** object

**Return**

The new **CTNumber** object
CTNumber::operator <, <=, >, >=, ==, !=

Syntax
CTBOOL operator < (const CTNumber& left, const CTNumber& right)
CTBOOL operator <= (const CTNumber& left, const CTNumber& right)
CTBOOL operator > (const CTNumber& left, const CTNumber& right)
CTBOOL operator >= (const CTNumber& left, const CTNumber& right)
CTBOOL operator == (const CTNumber& left, const CTNumber& right)
CTBOOL operator != (const CTNumber& left, const CTNumber& right)

Parameters
- **left** [in] The left CTNumber object to be compared
- **right** [in] The right CTNumber object to be compared

Description
These overloaded operators make comparisons between the CTNumber objects.

Return
The operators return YES or NO, depending on the result of the comparison.
CTNumber Methods

Constructor / Destructor
- **CTNumber()**: Creates a `CTNumber` object
- **~CTNumber()**: Destroys a `CTNumber` Object and resets all the dependent objects

Date Handling
- **IsZero()**: Indicate if a `CTNumber` value is zero
- **Zero()**: Set a `CTNumber` value to zero
- **SetNumber()**: Initializes the `CTNumber` object.
- **Round()**: Rounds the `CTNumber` to a specified number of decimal places
- **DecimalDigits()**: Retrieves the number of digits after the decimal point.
- **IntegralDigits()**: Retrieves the number of digits before the decimal point.
- **AsFloat()**: Converts a `CTNumber` object to double
- **AsLong()**: Converts a `CTNumber` object to long
- **AsNumber()**: Retrieves the C `CTNumber` type
- **AsMoney()**: Converts a `CTNumber` object to a `CTMoney` object
- **AsString()**: Converts a `CTNumber` object to a `CTString` object
- **AsBigInt()**: Converts a `CTNumber` to a big integer object.
- **AsCurrency()**: Converts `CTNumber` to a `CTCurrency` object.
### CTNumber::CTNumber

#### Syntax

- `CTNumber( )`
- `CTNumber(LONG value)`
- `CTNumber(CTFLOAT value)`
- `CTNumber(pCTNUMBER value)`
- `CTNumber(const CTMoney& value)`
- `CTNumber(const CTString& value)`
- `CTNumber(const CTBigint& value)`
- `CTNumber(const CTCurrency& value)`
- `CTNumber(const CTNumber& value)`

#### Parameters

- `value` [in] the value to be initially assigned to the new `CTNumber` object.

#### Description

This is the constructor for the `CTNumber` object.

#### See also

- `~CTNumber()`
CTNumber::~CTNumber

Syntax
~CTNumber

Parameters
This destructor has no parameters.

Description
This is the destructor for the CTNumber object.

See also
CTNumber()
CTNumber::AsBigInt

Syntax
CTBigInt AsBigInt( ) const

Parameters
This method has no parameters.

Description
Convert CTNumber to a big integer object.

Return
AsBigInt() returns the converted CTNumber value as a big integer object.

See also
AsLong(), AsFloat()
CTNumber::AsCurrency

Syntax
CTCurrency AsCurrency( ) const

Parameters
This method has no parameters.

Description
Convert CTNumber to CTCurrency.

Return
AsCurrency() returns the converted CTNumber value as a CTCurrency object.

See also
AsString()
CTNumber::AsFloat

Syntax

CTFLOAT AsFloat() const

Parameters

This method has no parameters.

Description

Converts a CTNumber object to double.

Return

AsFloat() returns the converted value in CTFLOAT format.

See also

AsLong(), AsBigInt()
CTNumber::AsLong

Syntax
LONG AsLong( ) const

Parameters
This method has no parameters.

Description
Converts a CTNumber object to LONG.

Return
AsLong() returns the converted value in LONG format.

See also
AsFloat(), AsBigint()
CTNumber::AsMoney

Syntax
CTMoney AsMoney( ) const

Parameters
This method has no parameters.

Description
Convert CTNumber to CTMoney.

Return
AsMoney() returns the converted CTNumber value as a CTMoney object.

See also
AsString()
CTNumber::AsNumber

Syntax
pCTNUMBER AsNumber( ) const

Parameters
This method has no parameters.

Description
Retrieve the C CTNumber type.

Return
AsNumber() returns a CTNumber object.

See also
AsLong(), AsFloat()
CTNumber::AsString

Syntax
CString AsString( ) const

Parameters
This method has no parameters.

Description
Converts a CTNumber object to a CTString object.

Return
AsString() returns a CTString object with the CTNumber value.

See also
AsBigint()
CTNumber::DecimalDigits

Syntax
NINT DecimalDigits( ) const

Parameters
This method has no parameters.

Description
DecimalDigits() gets the number of digits after the decimal point.

Return
DecimalDigits() returns the number of digits after the decimal point.

See also
IntegralDigits(), Round()
CTNumber::IntegralDigits

Syntax
NINT IntegralDigits( )  const

Parameters
This method has no parameters.

Description
IntegralDigits() gets the number of digits before the decimal point.

Return
IntegralDigits() returns the number of digits before the decimal point.

See also
DecimalDigits(), Round()
CTNumber::IsZero

Syntax

CTBOOL IsZero() const

Parameters

This method has no parameters.

Description

IsZero() indicates if a CTNumber value is zero.

Return

IsZero() returns YES if the CTNumber value is zero, NO otherwise.

See also

Zero()
CTNumber::Round

Syntax

CTNumber Round (NINT scale) const

Parameters

- `scale` [in] the number of decimal places.

Description

Rounds a number to a specified number of decimal places.

Return

Round returns the rounded number.

See also

IntegralDigits(), DecimalDigits()
CTNumber::SetNumber

Syntax
void SetNumber(LONG value)
void SetNumber(CTFLOAT value)
void SetNumber(const CTMoney& value)
void SetNumber(const CTString& value)
void SetNumber(const CTBigint& value)
void SetNumber(const CTCurrency& value);
void SetNumber(const CTNumber& value);
void SetNumber(pCTNUMBER value);

Parameters
- value [in] The long, double, CTBigint, CTMoney, CTCurrency, CTNumber or CTString value or object to be converted and assigned to the new CTNumber object

Description
Initializes the CTNumber object.

Return
None.

See also
Zero()
CTNumber::Zero

Syntax
void Zero

Parameters
This method has no parameters.

Description
Zero() sets a CTNumber value to zero.

Return
None.

See also
IsZero()
4.14 CTRecord Class

class CTRecord

Description
The CTRecord class deals with the record concept. The CTRecord object is the "atomic" or basic element of the database system, in the sense that it has the smallest piece of integrated information. It represents the information in the data structure defined in the table. Multiple records may belong to one table, but a record belongs to only one table. The records are several times defined as the "rows" in the tables.

See also
CTBase, CTDatabase, CTTable

Preconditions
In order to deal with the records, the user must logon to a Session, be connected to a Database, and have a Table open.
CTRecord Methods

Constructor / Destructor
- **CTRecord()** Creates a CTRecord object
- **~CTRecord()** Destroys a CTRecord Object and resets all the dependent objects

Record Search
- **First()**: Moves to the first record in the table
- **Last()**: Moves to the last record in the table.
- **Next()**: Moves to the next record in the table.
- **Prev()**: Moves to the previous record in the table.
- **Find()**: Looks for the record in the table.
- **FindRowid()**: Verifies if a record at a given rowid exist.
- **FindTarget()**: Looks for one specified record in the table.
- **GetRecordCount()**: Retrieves the number of records in table
- **GetRowid()**: Retrieves the record’s rowid value

Record Buffers
- **LockRecord()**: Locks the current record.
- **UnlockRecord()**: Unlocks the current record.
- **Clear()**: Clears the record buffer
- **Read()**: Reads the current record
- **Write()**: Writes the record buffer to disk
- **Delete()**: Deletes the current record from table
- **Reset()**: Resets the record buffer to its initial condition
- **IsNew()**: Indicates if a record buffer is cleared
- **IsEdited()**: Indicates if a record buffer has been modified
- **SetNew()**: Sets the new record flag
- **SetEdited()**: Sets the change record flag
- **GetRecordBuffer()**: Retrieves the record buffer object
- **GetRecordPos()**: Retrieves the current record offset position
- **SetRecordPos()**: Sets the current record offset position.
- **SetRecordOffset()**: Updates the record offset.
- **SeekRecord()**: Moves the record to position pointed by offset
- **GetRecordLength()**: Retrieves the record length.
- **GetRecordSize()**: Retrieves the allocated record size.
- **AtPercentile()**: Find a record located at about a given percentile value.
- **EstimateSpan()**: Estimate the approximate number of records between two key values.

Record Sets
- **RecordSetOn()**: Activates record sets.
- **RecordSetOff()**: Disactivates record sets.

### Field Handling
- **IsNullField()**: Indicates if the field pointed by field number or name is a null field.
- **GetFieldName()**: Retrieves the field name based on the field number
- **GetFieldType()**: Retrieves the field type based on the field number or name
- **GetFieldAsBool()**: Retrieves the field contents as a boolean value
- **GetFieldAsSigned()**: Retrieves the field contents as a signed value
- **GetFieldAsUnsigned()**: Retrieves the field contents as an unsigned value
- **GetFieldAsDate()**: Retrieves the field contents as a date object
- **GetFieldAsTime()**: Retrieves the field contents as a time object
- **GetFieldAsDateTime()**: Retrieves the field contents as a date and time object
- **GetFieldAsMoney()**: Retrieves the field contents as a money object
- **GetFieldAsFloat()**: Retrieves the field contents as a float value
- **GetFieldAsString()**: Retrieves the field contents as a string object
- **GetFieldAsBlob()**: Retrieves the field contents as a blob object
- **GetFieldAsBigint()**: Retrieves the field contents as a big integer (64-bit)
- **GetFieldAsCurrency()**: Retrieves the field contents as a `CTCurrency` object (64-bit)
- **GetFieldAsNumber()**: Retrieves the field contents as a `CTNumber` object
- **SetFieldAsBool()**: Sets the field contents as a boolean value
- **SetFieldAsSigned()**: Sets the field contents as a signed value
- **SetFieldAsUnsigned()**: Sets the field contents as an unsigned value
- **SetFieldAsDate()**: Sets the field contents as a date object
- **SetFieldAsTime()**: Sets the field contents as a time object
- **SetFieldAsDateTime()**: Sets the field contents as a date and time object
- **SetFieldAsMoney()**: Sets the field contents as a money object
- **SetFieldAsFloat()**: Sets the field contents as a float value
- **SetFieldAsString()**: Sets the field contents as a string object
- **SetFieldAsBlob()**: Sets the field contents as a blob object
- **SetFieldAsBigint()**: Sets the field contents as a big integer (64-bit)
- **SetFieldAsCurrency()**: Sets the field contents as a `CTCurrency` object (64-bit)
- **SetFieldAsNumber()**: Sets the field contents as a `CTNumber` object
- **GetFieldAsChar()**: Retrieves the field contents as a char value
- **GetFieldAsByte()**: Retrieves the field contents as a byte (one-byte unsigned NINT) value
- **GetFieldAsShort()**: Retrieves the field contents as a COUNT (two-bytes signed NINT) value
- **GetFieldAsWord()**: Retrieves the field contents as a UCOUNT (unsigned two-byte integer) value
- **SetFieldAsChar()**: Sets the field contents as a char value
- **SetFieldAsByte()**: Sets the field contents as a byte (one-byte unsigned NINT) value
- **SetFieldAsShort()**: Sets the field contents as a COUNT (two-bytes signed NINT) value
- **SetFieldAsWord()**: Sets the field contents as a UCOUNT (unsigned two-byte integer) value
- **IsVariableField()**: Indicates if a field is in the variable portion of a record
- **GetFieldLength()**: Retrieves the actual field data length
- **GetFieldSize()**: Retrieves the defined field size
- **GetFieldOffset()**: Retrieves the field offset in the record
- **GetFieldAddress()**: Retrieves the field address in the record buffer
- **GetFieldByName()**: Retrieves the field number based on the field name
- **ClearField()**: Clears the contents of a field.

**Index Handling**
- **GetDefaultIndex()**: Retrieves the record default index number
- **GetDefaultIndexName()**: Retrieves the default record index name.
- **SetDefaultIndex()**: Sets the new record default index number.

**Other Handling**
- **SwitchContext()**: Force a c-tree Plus ISAM context switch.
CTRecord::CTRecord

Syntax
CTRecord(const CTTable& Handle)
CTRecord(const CTTable* Handle)
CTRecord(const CTRecord& Handle)
CTRecord(const CTRecord* Handle)

Parameters
Handle [in] The Table or Record Object. If the prototype CTRecord(const CTTable& Handle) or CTRecord(const CTTable* Handle) is used, Handle is the Table Object. If the prototype CTRecord(const CTRecord & Handle) or CTRecord(const CTRecord * Handle) is used, Handle is the Record Object. The CTRecord object being created is a child of the CTTable or CTRecord object.

Description
This is the constructor for the CTRecord class.

See also
~CTRecord()
CTRecord::~CTRecord

Syntax

~CTRecord

Parameters

This destructor has no parameters.

Description

This is the destructor for the CTRecord class. It frees the allocated memory to the Record.

See also

CTRecord()
CTRecord::AtPercentile

Find a record located at about the given percentile value.

Declaration

void CTRecord::AtPercentile(NINT percent);

Description

CTRecord::AtPercentile() reads the record located at, approximately, the given percentile value.

Percent indicates the percentile value. The valid values for percent are from 0 to 100, indicating 0% to 100%. CTRecord::AtPercentile() return CTDBRET_OK on success.

The record is located using the record handle current index. You may select a new current index by calling the CTRecord::SetDefaultIndex() method. The table must have at least one index to be able to use this function.

The record returned is an approximation location indicated by the percentile value passed to CTRecord::AtPercentile().

CTRecord::AtPercentile(), which is based on c-tree low level function KeyAtPercentile(), and it is very efficient since it does not traverse all of the key values in order to determine the record located at the specified percentile. However, CTRecord::AtPercentile() is only an approximation since it assumes that key values are uniformly distributed among all of the b-tree leaf nodes.

CTRecord::AtPercentile() may be used to support scroll bar positioning, found in many GUI windowing environments, in the cases when the position must be maintained in key sequential order.

Return

CTRecord::AtPercentile() throws a CTException exception object if no record cannot be located.

See "c-treeDB Errors and Return Values" (page 881) for a complete listing of valid c-treeDB error codes and return values.

Example

pRec->Clear
pRec->AtPercentile(40);

See also

CTIndex::SetDefault()
CTRecord::BatchLoaded

Retrieves the number of batch records loaded into batch buffer.

**Declaration**

```cpp
LONG CTRecord::BatchLoaded();
```

**Description**

`CTRecord::BatchLoaded()` retrieves the number of batch records loaded into batch buffer for `CTBATCH_GET`, `CTBATCH_RANGE` or a `CTBATCH_PHYS` operation. This is the number of records that are ready to be retrieved by the `CTRecord::NextBatch()` function.

**Return**

`CTRecord::BatchLoaded()` returns the number of records ready for retrieval. In case of an error, `CTRecord::BatchLoaded()` throws an exception.

**Example**

```cpp
try
{
    // set the partial target key
    hRecord.Clear();
    hRecord.SetFieldAsSigned("Invoice", Invoice);
    // set the batch operation
    hRecord.SetBatch(CTBATCH_GET, sizeof(Invoice), 0);
    // show how many records are ready
    printf("%d records are ready\n", hRecord.BatchLoaded());
}
catch (CTException& err)
{
    printf("Batch failed with error %d\n", hRecord.GetError());
}
```

**See Also**

`CTRecord::BatchLocked()`, `CTRecord::BatchMode()`, `CTRecord::BatchTotal()`, `CTRecord::EndBatch()`, `CTRecord::InsertBatch()`, `CTRecord::IsBatchActive()`, `CTRecord::NextBatch()`, `CTRecord::SetBatch()`
**CTRecord::BatchLocked**

Retrieves the number of locked batch records.

**Declaration**

```cpp
LONG CTRecord::BatchLocked();
```

**Description**

`CTRecord::BatchLocked()` retrieves the number of records locked during a `CTBATCH_GET`, `CTBATH_RANGE` or `CTBATCH_PHYS` operation. If `CTBATCH_LOCK_READ` or `CTBATCH_LOCK_WRITE` are specified in the batch mode, the `CTRecord::BatchLocked()` method returns the total number of records locked. If `CTBATCH_LOCK_ONE` is specified, or if either the `CTBATCH_LOCK_READ` or `CTBATCH_LOCK_WRITE` modes are not specified, `CTRecord::BatchLocked()` returns zero.

**Return**

`CTRecord::BatchLocked()` returns the number of locked records. In case of an error, `CTRecord::BatchLocked()` throws a `CTException` exception.

**Example**

```cpp
try {
    // set the partial target key
    hRecord.Clear();
    hRecord.SetFieldAsSigned("Invoice", Invoice);
    // set the batch operation
    hRecord.SetBatch(CTBATCH_GET, sizeof(Invoice), 0);
    // show how many records are ready
    printf("%d records are locked\n", hRecord.BatchLocked());
} catch (CTException& err) {
    printf("Batch failed with error %d\n", hRecord.GetError());
}
```

**See Also**

`TRecord::BatchLoaded()`, `CTRecord::BatchMode()`, `CTRecord::BatchTotal()`, `CTRecord::EndBatch()`, `CTRecord::InsertBatch()`, `CTRecord::IsBatchActive()`, `CTRecord::NextBatch()`, `CTRecord::SetBatch()`
**CTRecord::BatchMode**

Retrieves the current batch mode.

**Declaration**

```cpp
CTBATCH_MODE CTRecord::BatchMode();
```

**Description**

`CTRecord::BatchMode()` retrieves the current batch mode. The batch mode is set by calling the `CTRecord::SetBatch()` method. If a batch operation is not active, `CTRecord::BatchMode()` returns `CTBATCH_NONE`.

**Return**

Returns the current batch mode or `CTBATCH_NONE` if no batch operation is currently active.

**Example**

```cpp
// check if a batch operation is in progress
if (hRecord.BatchMode() != CTBATCH_NONE)
    printf("Batch operation is underway\n");
else
    printf("No batch operations\n");
```

**See Also**

- `TRecord::BatchLoaded()`, `CTRecord::BatchLocked()`, `CTRecord::BatchTotal()`, `CTRecord::EndBatch()`, `CTRecord::InsertBatch()`, `CTRecord::IsBatchActive()`, `CTRecord::NextBatch()`, `CTRecord::SetBatch()`
CTRecord::BatchTotal

Retrieves the total number of records affected by a batch retrieval operation.

Declaration
LONG CTRecord::BatchTotal();

Description
CTRecord::BatchTotal() retrieves the total number of records selected by a batch retrieval operation. If a batch operation is not active, CTRecord::BatchTotal() returns zero.

Return
Returns the total number of records selected by a batch retrieval operation. In case of an error, CTRecord::BatchTotal() throws a CTException.

Example
try
{
    // set the partial target key
    hRecord.Clear();
    hRecord.SetFieldAsSigned("Invoice", Invoice);
    // set the batch operation
    hRecord.SetBatch(CTBATCH_GET, sizeof(Invoice), 0);
    // show how many records are ready
    printf("%d records total\n", hRecord.BatchLocked());
}
catch (CTException& err)
{
    printf("Batch failed with error %d\n", hRecord.GetError());
}

See Also
TRecord::BatchLoaded(), CTRecord::BatchLocked(), CTRecord::BatchMode(), CTRecord::EndBatch(), CTRecord::InsertBatch(), CTRecord::IsBatchActive(), CTRecord::NextBatch(), CTRecord::SetBatch()
**CTRecord::BuildTargetKey**

**Syntax**

```c
void BuildTargetKey(CTFIND_MODE FindMode, pVOID targetkey, pVRLEN targetlen);
```

**Parameters**

- `FindMode` [in] The find modes are: `CTFIND_EQ`, `CTFIND_LT`, `CTFIND_LE`, `CTFIND_GT` and `CTFIND_GE`.
- `targetkey` [out] target key buffer
- `targetlen` [in/out] target key length. Before calling `ctdbBuildTargetKey()` set `targetlen` with the size of `targetkey` buffer.

**Description**

Build a target key based on data in record buffer

**Return**

None.

**See also**

- `FindTarget()`
CTRecord::Clear

Syntax
void Clear( )

Parameters
This method has no parameters.

Description
Clears the record buffer. Following this, c-treeDB no longer has a current record.

Return
None.

Example
Rec1 = new CTRecord(pTable);
Rec1->Clear();
Rec1->SetFieldAsString("name", "Peter Thorn");
Rec1->SetFieldAsLong("age", 35);
Rec1->SetFieldAsString("phone", "555-55555");
Rec1->Write();

See also
Read(), Write(), Delete(), Reset(), Lock()
CTRecord::ClearField

Syntax
void ClearField(const NINT FieldNbr)
void ClearField(const CTString& FieldName)

Parameters
- FieldNbr [in] The field number.
- FieldName [in] The field name.

Description
Clears the contents of a field.

Return
None.

See also
Clear(), Reset()
CTRecord::Delete

Syntax
void Delete();

Parameters
This method has no parameters.

Description
Deletes the current record from table. The record must be locked with one of the write locks before it is deleted.

Return
None.

Example
pRec->Clear();
pRec->SetDefaultIndex("name");
pRec->SetFieldAsString("name", "Peter Thorn");
if(pRec->Find(CTFIND_EQ))
{
    pRec->LockRecord(CTLOCK_WRITE_BLOCK);
    pRec->Delete();
}

See also
Clear(), Reset(), Read(), Write(), LockRecord(), CTBase::Lock()
CTRecord::EndBatch

Terminates or cancels a batch operation.

Declaration

void CTRecord::EndBatch();

Description

A batch operation must be terminated by calling the CTRecord::EndBatch() function. Once a batch operation is started, by calling CTRecord::SetBatch(), no other batch operation is allowed to start until the current batch operation is terminated.

When performing batch retrieval operations, you may cancel the batch operation before retrieving all the records by calling CTRecord::EndBatch().

If the batch operation is a CTBATCH_RANGE then you must also call the CTRecord::RangeOff() function to terminate the index range used for the batch operation.

In case of errors, CTRecord::EndBatch() throws a CTException.

Return

void

Example

try
{
    // set the partial target key
    hRecord.Clear();
    hRecord.SetFieldAsSigned("Invoice", Invoice);
    // set the batch operation
    hRecord.SetBatch(CTBATCH_DEL, sizeof(Invoice), 0);
    // end the batch operation
    hRecord.EndBatch();
}

catch (CTException& err)
{
    printf("Batch failed with error %d\n", hRecord.GetError());
}

See Also

TRecord::BatchLoaded(), CTRecord::BatchLocked(), CTRecord::BatchMode(), CTRecord::BatchTotal(), CTRecord::InsertBatch(), CTRecord::IsBatchActive(), CTRecord::NextBatch(), CTRecord::SetBatch()
CTRecord::EstimateSpan

Estimate an approximate number of records between two key target values.

**Declaration**

LONG CTRecord::EstimateSpan(pVOID key1, pVOID key2);

**Description**

- *key1* and *key2* are two key target values used to obtain the estimated number of records.

If CTRecord::EstimateSpan() returns 0, use CTRecord::GetError() function to retrieve the error code. If CTRecord::EstimateSpan() returns 0 and CTRecord::GetError() returns CTDBRET_OK then there are no records between the two key values supplied.

The estimation is based on the record handle current index. The current index may be changed by calling CTIndex::SetDefault(). The table must have at least one index to be able to use this function.

CTRecord::EstimateSpan(), which is based on the c-tree low level function CTRecord::EstimateSpan(), does not traverse the index to compute the values. Instead, it makes approximately ten calls to the c-tree low level function KeyAtPercentile() to determine the relative location of the target values.

The key target values used by CTRecord::EstimateSpan() can be created using CTRecord::BuildTargetKey().

**Return**

CTRecord::EstimateSpan() returns an estimate of records between *key1* and *key2*. Call the CTRecord::GetError() method to check for error conditions.

**Example**

LONG Estimate(CTRecord& Handle, NINT index)
{
    LONG Retval = 0;
    TEXT key1[16], key2[16];
    VRLEN klen;

    try
    {
        // set the default index
        Handle->SetDefaultIndex(index);

        // load the first record
        Handle->First();

        // build the target key for the first record
        klen = sizeof(key1);
        Handle->BuildTargetKey(CTFIND_EQ, key1, &klen);

        // load the last record
        Handle->Last();

        // build the target key for the last record
        klen = sizeof(key2);
        Handle->BuildTargetKey(CTFIND_EQ, key2, &klen);
    }
}
// get the estimated span
Retval = Handle->EstimateSpan(key1, key2);
if (Retval > 0)
    Retval--;
}

See also
CTIndex::SetDefault(), CTRecord::BuildTargetKey()
CTRecord::Find

Syntax

CTBOOL Find (CTFIND_MODE FindMode)

Parameters

- **FindMode** [in] The find mode for the search. Valid values for the find mode are shown in the "Find Modes" (page 119).
  Notice that when the FindMode **CTFIND_EQ** is used, all segments that compose the index must be fulfilled, and also the index cannot allow duplicates.

Description

Looks for the record in the table.

Return

Find return YES if the record is found, NO otherwise.

Example

```c
pRec->Clear();
pRec->SetDefaultIndex("name");
pRec->SetFieldAsString("name", "Peter Thorn");
if(pRec->Find(CTFIND_EQ))
{
    pRec->LockRecord(CTLOCK_WRITE_BLOCK);
    pRec->Delete();
}
```

See also

First(), Next(), Prev(), Last(), FindTarget()
CTRecord::FindRowid

Syntax
CTBOOL FindRowid(CTROWID rowid, CTFIND_MODE FindMode)

Parameters
- rowid [in] Rowid of record being sought
- FindMode [in] The find mode for the search. Valid values for the find mode are shown in the “Find Modes” (page 119).
  Notice that when the FindMode CTFIND_EQ is used, all segments that compose the index must be fulfilled, and also the index cannot allow duplicates.

Description
Verifies if a record exist at a given rowid value. To retrieve the record rowid, use GetRowid().
One table has the rowid record enabled by default at creation time. To verify if one table has support to rowid, use CTTable::HasRowid().

Return
FindRowid() returns YES if the record is located, NO otherwise.

See also
GetRowid(), CTTable::HasRowid()
CTRecord::FindTarget

Syntax
CTBOOL FindTarget(pVOID target, CTFIND_MODE FindMode)

Parameters
- `target` [in] The target key to look for in the table. `target` must be a properly transformed key value (use `BuildTargetKey()` if necessary).
- `FindMode` [in] The find mode for the search. Valid values for the find mode are shown in the "Find Modes" (page 119).
  Notice that when the `FindMode` CTFIND_EQ is used, all segments that compose the index must be fulfilled, and also the index cannot allow duplicates.

Note: Prior to c-treeACE V9.1 target must NOT be transformed.

Description
Looks for one specified record in the table.

Return
`FindTarget()` return YES if the record is found, NO otherwise.

See also
`First()`, `Next()`, `Prev()`, `Last()`, `Find()`
CTRecord::First

Syntax

CTBOOL First( )

Parameters

This method has no parameters.

Description

Moves to first record in table.

Return

First returns YES if the record is found, NO otherwise.

Example

pRec->First();
do {
    printf(\"\nName: %s\", pRec ->
       GetFieldAsString("name").c_str());
}while (pRec->Next());

See also

Next(), Prev(), Last(), Find()
CTRecord::GetDefaultIndex

Syntax

NINT GetDefaultIndex( )

Parameters

This method has no parameters.

Description

Retrieves the default record index number.

Return

GetDefaultIndex() returns the default index number.

- `CTDB_DATA_IDXNO`: indicates a physical data table traversal without using any indices.

Example

i = pRec->GetDefaultIndex();
if (i == 1)
{
    pRec->Clear();
    pRec->First();
}

See also

SetDefaultIndex(), GetDefaultIndexName()
CTRecord::GetDefaultIndexName

Syntax

CTString GetDefaultIndexName( )

Parameters

This method has no parameters.

Description

Retrieves the default record index name.

Return

GetDefaultIndexName() returns the default index name.

Example

if ("name" == pRec->GetDefaultIndexName())
{
    pRec->Clear();
    pRec->SetFieldAsString("name", "Peter Thorn");
    pRec->Find(CTFIND_EQ);
}

See also

GetDefaultIndex(), SetDefaultIndex()
CTRecord::GetErrorIndex

Declaration
NINT CTRecord::GetErrorIndex();

Description
Retrieves the index number that cause a record insert or record update operation to fail. This function should only be called after a Write() call fails. The error index number value is maintained until the next call to Write().

Return Values
A number from 0 to n to indicate which index caused the error. A value of 0 represents the first index, 1 represents the second index, and so on. If the index number can not be obtained, GetErrorIndex() returns -1.

See Also
CTRecord::Write()
**CTRecord::GetFieldAddress**

**Syntax**

```cpp
pVOID GetFieldAddress(NINT FieldNbr)
pVOID GetFieldAddress(const CTString& FieldName)
```

**Parameters**

- `FieldNbr [in]` The field number.
- `FieldName [in]` The field name.

**Description**

Retrieves the field address in the record buffer.

**Return**

`GetFieldAddress()` returns the field address.

**See also**

`GetFieldSize()`, `GetFieldName()`, `GetFieldType()`, `GetFieldOffset()`
CTRecord::GetFieldAsBigint

Syntax
CTBigint GetFieldAsBigint(NINT FieldNumber)
CTBigint GetFieldAsBigint(const CTString& FieldName)

Parameters
- FieldNumber [in] The field number.
- FieldName [in] The field name.

Description
Retrieves the field contents as a big integer (64-bit).

Return
GetFieldAsBigint() returns the contents as a big integer value.

See also
GetFieldAsBool(), GetFieldAsByte(), GetFieldAsBlob(), GetFieldAsChar(),
GetFieldAsDate(), GetFieldAsTime(), GetFieldAsDateTime(), GetFieldAsMoney(),
GetFieldAsShort(), GetFieldAsFloat(), GetFieldAsSigned(), GetFieldAsUnsigned(),
GetFieldAsWord(), GetFieldAsString(), GetFieldAsCurrency(), GetFieldAsBigint(),
GetFieldAsNumber()
CTRecord::GetFieldAsBlob

Syntax
void GetFieldAsBlob(NINT FieldNumber, CTBlob& value)
void GetFieldAsBlob(const CTString& FieldName, CTBlob& value)

Parameters
- FieldNumber [in] The field number.
- value [out] The Blob object, returned by the method.
- FieldName [in] The field name.

Description
Retrieves the field contents as a CTBlob object

Return
None.

See also
GetFieldAsBool(), GetFieldAsByte(), GetFieldAsChar(), GetFieldAsDate(),
GetFieldAsTime(), GetFieldAsDateTime(), GetFieldAsMoney(), GetFieldAsShort(),
GetFieldAsFloat(), GetFieldAsSigned(), GetFieldAsUnsigned(), GetFieldAsWord(),
GetFieldAsString(), GetFieldAsCurrency(), GetFieldAsBigint(), GetFieldAsNumber()
CTRecord::GetFieldAsBool

Syntax
CTBOOL GetFieldAsBool(NINT FieldNumber)
CTBOOL GetFieldAsBool(const CTString& FieldName)

Parameters
- **FieldNumber** [in] The field number.
- **FieldName** [in] The field name.

Description
Retrieves the field contents as a boolean value

Return
GetFieldAsBool() returns the contents as a boolean value.

See also
GetFieldAsByte(), GetFieldAsBlob(), GetFieldAsChar(), GetFieldAsDate(),
GetFieldAsTime(), GetFieldAsDateTime(), GetFieldAsMoney(), GetFieldAsShort(),
GetFieldAsFloat(), GetFieldAsSigned(), GetFieldAsUnsigned(), GetFieldAsWord(),
GetFieldAsString(), GetFieldAsCurrency(), GetFieldAsBigInt(), GetFieldAsNumber()
CTRecord::GetFieldAsByte

Syntax
UTEXT GetFieldAsByte(NINT FieldNumber)
UTEXT GetFieldAsByte(const CTString& FieldName)

Parameters
- FieldNumber [in] The field number.
- FieldName [in] The field name.

Description
Retrieves the field contents as a UTEXT value (unsigned one-byte integer).

Return
GetFieldAsByte() returns the contents as a UTEXT value.

See also
GetFieldAsBool(), GetFieldAsBlob(), GetFieldAsChar(), GetFieldAsDate(),
GetFieldAsTime(), GetFieldAsDateTime(), GetFieldAsMoney(), GetFieldAsShort(),
GetFieldAsFloat(), GetFieldAsSigned(), GetFieldAsUnsigned(), GetFieldAsWord(),
GetFieldAsString(), GetFieldAsCurrency(), GetFieldAsBigInt(), GetFieldAsNumber()
CTRecord::GetFieldAsChar

Syntax
TEXT GetFieldAsChar(NINT FieldNumber)
TEXT GetFieldAsChar(const CTString& FieldName)

Parameters
- **FieldNumber** [in] The field number.
- **FieldName** [in] The field name.

Description
Retrieves the field contents as a char value.

Return
GetFieldAsChar() returns the contents as a TEXT value.

See also
GetFieldAsBool(), GetFieldAsByte(), GetFieldAsBlob(), GetFieldAsDate(),
GetFieldAsTime(), GetFieldAsDateTime(), GetFieldAsMoney(), GetFieldAsShort(),
GetFieldAsFloat(), GetFieldAsSigned(), GetFieldAsUnsigned(), GetFieldAsWord(),
GetFieldAsString(), GetFieldAsCurrency(), GetFieldAsBigInt(), GetFieldAsNumber()
CTRecord::GetFieldAsCurrency

Syntax

CTCurrency GetFieldAsCurrency(NINT FieldNumber)
CTCurrency GetFieldAsCurrency(const CTString& FieldName)

Parameters

- FieldNumber [in] The field number.
- FieldName [in] The field name.

Description

Retrieves the field contents as a CTCurrency object.

Return

GetFieldAsCurrency() returns the contents as a CTCurrency object.

See also

GetFieldAsBool(), GetFieldAsByte(), GetFieldAsBlob(), GetFieldAsChar(), GetFieldAsDate(), GetFieldAsTime(), GetFieldAsDateTime(), GetFieldAsShort(), GetFieldAsFloat(), GetFieldAsSigned(), GetFieldAsUnsigned(), GetFieldAsWord(), GetFieldAsString(), GetFieldAsMoney(), GetFieldAsBigInt(), GetFieldAsNumber()
CTRecord::GetFieldAsDate

Syntax

CTDate GetFieldAsDate(NINT FieldNumber)
CTDate GetFieldAsDate(const CTString& FieldName)

Parameters

- FieldNumber [in] The field number.
- FieldName [in] The field name.

Description

Retrieves the field contents as a CTDDate object

Return

GetFieldAsDate() returns the contents as CTDDate object.

See also

GetFieldAsBool(), GetFieldAsByte(), GetFieldAsBlob(), GetFieldAsChar(), GetFieldAsTime(), GetFieldAsDateTime(), GetFieldAsMoney(), GetFieldAsShort(), GetFieldAsFloat(), GetFieldAsSigned(), GetFieldAsUnsigned(), GetFieldAsWord(), GetFieldAsString(), GetFieldAsCurrency(), GetFieldAsBigint(), GetFieldAsNumber()
**CTRecord::GetFieldAsDateTime**

**Syntax**

```c
CTDateTime GetFieldAsDateTime(NINT FieldNumber)
CTDateTime GetFieldAsDateTime(const CTString& FieldName)
```

**Parameters**

- `FieldNumber` [in] The field number.
- `FieldName` [in] The field name.

**Description**

Retrieves the field contents as a `CTDateTime` object.

**Return**

`GetFieldAsDateTime()` returns the contents as a `CTDateTime` object.

**See also**

- `GetFieldAsBool()`, `GetFieldAsByte()`, `GetFieldAsBlob()`, `GetFieldAsChar()`, `GetFieldAsDate()`, `GetFieldAsTime()`, `GetFieldAsMoney()`, `GetFieldAsShort()`, `GetFieldAsFloat()`, `GetFieldAsSigned()`, `GetFieldAsUnsigned()`, `GetFieldAsWord()`, `GetFieldAsString()`, `GetFieldAsCurrency()`, `GetFieldAsBigint()`, `GetFieldAsNumber()`
CTRecord::GetFieldAsFloat

Syntax
CTFLOAT GetFieldAsFloat(NINT FieldNumber)
CTFLOAT GetFieldAsFloat(const CString& FieldName)

Parameters
- FieldNumber [in] The field number.
- FieldName [in] The field name.

Description
Retrieves the field contents as a float value

Return
GetFieldAsFloat() returns the contents as a float value.

See also
GetFieldAsBool(), GetFieldAsByte(), GetFieldAsBlob(), GetFieldAsChar(), GetFieldAsDate(), GetFieldAsTime(), GetFieldAsDateTime(), GetFieldAsMoney(), GetFieldAsShort(), GetFieldAsSigned(), GetFieldAsUnsigned(), GetFieldAsWord(), GetFieldAsString(), GetFieldAsCurrency(), GetFieldAsBigint(), GetFieldAsNumber()
CTRecord::GetFieldAsMoney

Syntax
CTMoney GetFieldAsMoney(NINT FieldNumber)
CTMoney GetFieldAsMoney(const CTString& FieldName)

Parameters
- **FieldNumber** [in] The field number.
- **FieldName** [in] The field name.

Description
Retrieves the field contents as a *CTMoney* object.

Return
*GetFieldAsMoney()* returns the contents as a *CTMoney* object.

See also
*GetFieldAsBool()*, *GetFieldAsByte()*, *GetFieldAsBlob()*, *GetFieldAsChar()*,
*GetFieldAsDate()*, *GetFieldAsTime()*, *GetFieldAsDateTime()*, *GetFieldAsShort()*,
*GetFieldAsFloat()*, *GetFieldAsSigned()*, *GetFieldAsUnsigned()*, *GetFieldAsWord()*,
*GetFieldAsString()*, *GetFieldAsCurrency()*, *GetFieldAsBigint()*, *GetFieldAsNumber()*
**CTRecord::GetFieldAsNumber**

**Syntax**

```cpp
CTNumber GetFieldAsNumber(NINT FieldNumber)
CTNumber GetFieldAsNumber(const CTString& FieldName)
```

**Parameters**

- `FieldNumber` [in] The field number.
- `FieldName` [in] The field name.

**Description**

Retrieves the field contents as a `CTNumber` object.

**Return**

`GetFieldAsFloat()` returns the contents as a `CTNumber` object.

**See also**

- `GetFieldAsBool()`, `GetFieldAsByte()`, `GetFieldAsBlob()`, `GetFieldAsChar()`, `GetFieldAsDate()`, `GetFieldAsTime()`, `GetFieldAsDateTime()`, `GetFieldAsMoney()`, `GetFieldAsShort()`, `GetFieldAsFloat()`, `GetFieldAsSigned()`, `GetFieldAsUnsigned()`, `GetFieldAsWord()`, `GetFieldAsString()`, `GetFieldAsCurrency()`, `GetFieldAsBigint()`
CTRecord::GetFieldAsShort

Syntax
COUNT GetFieldAsShort(NINT FieldNumber)
COUNT GetFieldAsShort(const CTString& FieldName)

Parameters
- *FieldName* [in] The field name.
- *FieldNumber* [in] The field number.

Description
Retrieves the field contents as a 2-bytes signed integer (COUNT) value.

Return
GetFieldAsShort() returns the contents as a COUNT value.

See also
GetFieldAsBool(), GetFieldAsByte(), GetFieldAsBlob(), GetFieldAsChar(),
GetFieldAsDate(), GetFieldAsTime(), GetFieldAsDateTime(), GetFieldAsMoney(),
GetFieldAsFloat(), GetFieldAsSigned(), GetFieldAsUnsigned(), GetFieldAsWord(),
GetFieldAsString(), GetFieldAsCurrency(), GetFieldAsBigInt(), GetFieldAsNumber()
CTRecord::GetFieldAsSigned

Syntax
CTSHORT GetFieldAsSigned(NINT FieldNumber)
CTSHORT GetFieldAsSigned(const CTString& FieldName)

Parameters
- **FieldNumber** [in] The field number.
- **FieldName** [in] The field name.

Description
Retrieves the field contents as a signed value

Return
GetFieldAsSigned() returns the contents as a signed value.

See also
GetFieldAsBool(), GetFieldAsByte(), GetFieldAsBlob(), GetFieldAsChar(),
GetFieldAsDate(), GetFieldAsTime(), GetFieldAsDateTime(), GetFieldAsMoney(),
GetFieldAsShort(), GetFieldAsFloat(), GetFieldAsUnsigned(), GetFieldAsWord(),
GetFieldAsString(), GetFieldAsCurrency(), GetFieldAsBigInt(), GetFieldAsNumber()
CTRecord::GetFieldAsString

Syntax
CTString GetFieldAsString(NINT FieldNumber)
CTString GetFieldAsString(const CTString& FieldName)

Parameters
- **FieldNumber** [in] The field number.
- **FieldName** [in] The field name.

Description
Retrieves the field contents as a **CTString** object.

Return
**GetFieldAsString()** returns the contents as a **CTString** object.

See also
- GetFieldAsBool(), GetFieldAsByte(), GetFieldAsBlob(), GetFieldAsChar(),
  GetFieldAsDate(), GetFieldAsTime(), GetFieldAsDateTime(), GetFieldAsMoney(),
  GetFieldAsShort(), GetFieldAsFloat(), GetFieldAsSigned(), GetFieldAsUnsigned(),
  GetFieldAsWord(), GetFieldAsCurrency(), GetFieldAsBsignt(), GetFieldAsNumber()
CTRecord::GetFieldAsTime

Syntax

CTTime GetFieldAsTime(NINT FieldNumber)
CTTime GetFieldAsTime(const CTString& FieldName)

Parameters

- \textit{FieldNumber} [in] The field number.
- \textit{FieldName} [in] The field name.

Description

Retrieves the field contents as a \textit{CTTime} object

Return

\textbf{GetFieldAsTime()} returns the contents as a \textit{CTTime} object.

See also

\textbf{GetFieldAsBool()}, \textbf{GetFieldAsByte()}, \textbf{GetFieldAsBlob()}, \textbf{GetFieldAsChar()}, \textbf{GetFieldAsDate()}, \textbf{GetFieldAsDateTime()}, \textbf{GetFieldAsMoney()}, \textbf{GetFieldAsShort()}, \textbf{GetFieldAsFloat()}, \textbf{GetFieldAsSigned()}, \textbf{GetFieldAsUnsigned()}, \textbf{GetFieldAsWord()}, \textbf{GetFieldAsString()}, \textbf{GetFieldAsCurrency()}, \textbf{GetFieldAsBigint()}, \textbf{GetFieldAsNumber()}
CTRecord::GetFieldAsUnsigned

Syntax
CTUNSIGNED GetFieldAsUnsigned (NINT FieldNumber)
CTUNSIGNED GetFieldAsUnsigned(const CTString& FieldName)

Parameters
- FieldNumber [in] The field number.
- FieldName [in] The field name.

Description
Retrieves the field contents as an unsigned value

Return
GetFieldAsUnsigned() returns the contents as an unsigned value.

See also
GetFieldAsBool(), GetFieldAsByte(), GetFieldAsBlob(), GetFieldAsChar(), GetFieldAsDate(), GetFieldAsTime(), GetFieldAsDateTime(), GetFieldAsMoney(), GetFieldAsShort(), GetFieldAsFloat(), GetFieldAsSigned(), GetFieldAsWord(), GetFieldAsString(), GetFieldAsCurrency(), GetFieldAsBigint(), GetFieldAsNumber()
CTRecord::GetFieldAsUTF16

Retrieves the field data as a Unicode UTF-16 string.

**Declaration**

```c
void CTRecord::GetFieldAsUTF16(NINT FieldNumber, pWCHAR value, VRLEN size);
```

**Description**

`CTRecord::GetFieldAsUTF16()` retrieves the field data as a UNICODE UTF-16 string. If the underlying field type is not one of the UNICODE field types, the data is converted to UTF-16 strings. **FieldNumber** is the number of the field, value is a pointer to a wide (UTF-16) string buffer and size indicates the size in bytes of the string area.

**Return**

`void`

**Example**

```c
void CheckData(CTRecord& hRecord, const CTString& str, NINT val)
{
    WCHAR WStr[32];
    TEXT s[64];
    CTSIGNED t;

    hRecord.GetFieldAsUTF16(0, WStr, sizeof(WStr));
    ctdb_u16ToU8(WStr, s, sizeof(s));
    if (strcmp(s, str) != 0)
        printf("UNICODE field contents not the same written");

    t = hRecord.GetFieldAsSigned(1);
    if ((NINT)t != val)
        printf("integer field contents not the same written");
}
```

**See Also**

`SetFieldAsUTF16()`
**CTRecord::GetFieldAsWord**

**Syntax**

UCOUNT GetFieldAsWord(NINT FieldNumber)

UCOUNT GetFieldAsWord(const CTString& FieldName)

**Parameters**

- **FieldNumber** [in] The field number.
- **FieldName** [in] The field name.

**Description**

Retrieves the field contents as an unsigned two-bytes integer value (UCOUNT).

**Return**

GetFieldAsWord() returns the contents as an UCOUNT value.

**See also**

GetFieldAsBool(), GetFieldAsByte(), GetFieldAsBlob(), GetFieldAsChar(), GetFieldAsDate(), GetFieldAsTime(), GetFieldAsDateTime(), GetFieldAsMoney(), GetFieldAsShort(), GetFieldAsFloat(), GetFieldAsSigned(), GetFieldAsUnsigned(), GetFieldAsString(), GetFieldAsCurrency(), GetFieldAsBigint(), GetFieldAsNumber()
**CTRecord::GetFieldByName**

**Syntax**

NINT GetFieldByName(const CTString& FieldName)

**Parameters**

- *FieldName* [in] The field name.

**Description**

Retrieves the field number based on the field name.

**Return**

GetFieldByName() returns the field number.

**See also**

GetFieldName(), GetFieldLength(), GetFieldOffset(), GetFieldType(), GetFieldSize()
CTRecord::GetFieldLength

Syntax
VRLEN GetFieldLength(NINT FieldNbr)
VRLEN GetFieldLength(const CTString& FieldName)

Parameters
- FieldNbr [in] The field number.
- FieldName [in] The field name.

Description
Retrieves the actual field data length.

Return
GetFieldLength() returns the field length.

See also
GetFieldSize(), GetFieldName(), GetFieldType(), GetFieldOffset()
CTRecord::GetFieldName

Syntax
CTString GetFieldName(NINT FieldNumber)

Parameters
- **FieldNumber [in]** The field number.

Description
Retrieves the field name based on the field number.

Return
GetFieldName() returns the field name.

Example
field_name=pRec->GetFieldName(1);

See also
GetFieldType(), GetFieldOffset()
CTRecord::GetFieldOffset

Syntax

VRLEN GetFieldOffset(NINT FieldNbr)
VRLEN GetFieldOffset(const CTString& FieldName)

Parameters

- **FieldNbr** [in] The field number.
- **FieldName** [in] The field name.

Description

Retrieves the field record offset.

Return

GetFieldOffset() returns the field offset.

See also

GetFieldLength(), GetFieldName(), GetFieldType(), GetFieldAddress()
**CTRecord::GetFieldSize**

**Syntax**

```
VRLEN GetFieldSize (NINT FieldNbr)
VRLEN GetFieldSize(const CTString& FieldName)
```

**Parameters**

- `FieldNbr [in]` The field number.
- `FieldName [in]` The field name.

**Description**

Retrieves the field defined size.

**Return**

`GetFieldSize()` returns the defined field size.

**See also**

`GetFieldLength()`, `GetFieldName()`, `GetFieldType()`, `GetFieldOffset()`
CTRecord::GetFieldType

Syntax

CTDBTYPE GetFieldType(NINT FieldNumber)
CTDBTYPE GetFieldType(const CTString& FieldName)

Parameters

- FieldNumber [in] The field number.
- FieldName [in] The field name.

Description

Retrieves the field type based on the field number or name.

Return

GetFieldType() returns the field type.

See also

GetFieldName(), GetFieldOffset()
CTRecord::GetFilter

Retrieves the current filter expression text.

**Declaration**

```cpp
void CTRecord::GetFilter(CTString& cndexpr);
```

**Description**

`CTRecord::GetFilter()` retrieves the current filter expression text. If no record filter is active for this record object, an empty string is returned.

**Return**

void

**Example**

```cpp
// if the record filter is active, retrieve the filter expression
if  (hRecord.IsFiltered())
{
    CTString expr;
    hRecord.GetFilter(expr);
    printf("Filter expression %s\n", expr.c_str());
}
```

**See Also**

`CTRecord::SetFilter()`, `CTRecord::IsFiltered()`
CTRecord::GetRecordBuffer

Syntax
pVOID GetRecordBuffer( )

Parameters
This method has no parameters.

Description
Retrieves the record buffer object

Return
GetRecordBuffer() returns the record buffer object.
CTRecord::GetRecordCount

Syntax

CTUINT64 GetRecordCount( )

Parameters

This method has no parameters.

Description

Retrieves the number of records in table. It may be used in conjunction with First and Next to retrieve all records in the table.

Return

GetRecordCount() returns the number of records in the table.

See also

First(), Next()
CTRecord::GetRecordKeyPos

Syntax
CTOFFSET CRecord::GetRecordKeyPos() const

Parameters
- None

Description
Retrieves the given record position in the default index.

Return
Position. The returned position is the Ordinal key position, not a file offset.

Throws a CException exception object if an error occurs.
CTRecord::GetRecordLength

Syntax
VRLEN GetRecordLength( ) const

Parameters
This method has no parameters.

Description
Retrieves the record length.

Return
GetRecordLength() returns the record length.

See also
GetRecordSize()
**Syntax**

`CTOFFSET GetRecordPos( ) const`

**Parameters**

This method has no parameters.

**Description**

Retrieves the current record offset position

**Return**

None.

**See also**

`SetRecordOffset()`
CTRecord::GetRecordSize

Syntax
VRLEN GetRecordSize( ) const

Parameters
This method has no parameters.

Description
Retrieves the allocated record size.

Return
GetRecordSize() returns the allocated record size.

See also
GetRecordLength()
CTRecord::GetRowid

Syntax

CTROWID GetRowid( )

Parameters

This method has no parameters.

Description

Retrieves the record’s rowid value

Return

GetRowid() returns the rowid for the record.

See also

FindRowid(), CTTable::HasRowid()
CTRecord::InsertBatch

Inserts a new record into a batch buffer.

Declaration
void CTRecord::InsertBatch();

Description
Inserts a new record into a batch buffer maintained internally by c-treeDB. When the batch buffer fills up, the group of records stored in the batch buffer are inserted into the table. If CTRecord::EndBatch() is called and the batch buffer still contains records, a new insert record operation is performed for the remaining records before the batch operation is terminated.

For transaction controlled files, the batch insertion operation is treated as one all or nothing operation. If no explicit transaction is started, each insertion of records will start and end its own transaction. Even if an explicit transaction is started, each insertion operation is treated independently through safe points.

Note: currently, all record insertion operations will not perform any conversion of record images, key values and record position for heterogeneous client/server implementations.

The following steps must be taken to perform a batch insert record operation:

1. Call CTRecord::SetBatch() function, with CTBATCH_INS mode, to insert a group of records.
   For each record to be inserted perform the following operations:
   a. Call CTRecord::Clear() to clear a record buffer
   b. For each field in the record call one of the CTRecord::SetFieldAs...() functions to set the field data.
   c. Call CTRecord::InsertBatch() to insert the record into the batch buffer.
2. Call CTRecord::EndBatch() to indicate that no more records will be inserted.

In case of errors, CTRecord::InsertBatch() throws a CTException.

Example

try
{
    // set the batch operation
    hRecord.SetBatch(CTBATCH_INS, 0, 0);
    // prepare the first record
    hRecord.Clear();
    hRecord.SetFieldAsSigned("Invoice", Invoice);// invoice
    hRecord.SetFieldAsSigned("ItemNbr", 1);      // invoice item
    hRecord.SetFieldAsSigned("Quantity", 100);   // item quantity
    hRecord.SetFieldAsSigned("ItemCode", 1001);  // item code
    hRecord.InsertBatch();                      // insert
    // prepare the second record
    hRecord.Clear();
    hRecord.SetFieldAsSigned("Invoice", Invoice);// invoice
    hRecord.SetFieldAsSigned("ItemNbr", 2);      // invoice item
hRecord.SetFieldAsSigned("Quantity", 200);  // item quantity
hRecord.SetFieldAsSigned("ItemCode", 1002);  // item code
hRecord.InsertBatch();                       // insert
// terminate the batch operation
hRecord.EndBatch();
}
catch (CTException& err)
{
    printf("Batch failed with error %d\n", hRecord.GetError());
}

See Also
CTRecord::BatchLoaded(), CTRecord::BatchLocked(), CTRecord::BatchMode(),
CTRecord::BatchTotal(), CTRecord::EndBatch(), CTRecord::IsBatchActive(),
CTRecord::NextBatch(), CTRecord::SetBatch()
CTRecord::IsBatchActive

Indicates if a batch operation is under way or not.

Declaration

CTBOOL CTRecord::IsBatchActive();

Description

CTRecord::IsBatchActive() indicates if a batch operation is active or not. This is equivalent to CTRecord::BatchMode() returning CTBATCH_NONE.

Return

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO</td>
<td>No batch operation is under way.</td>
</tr>
<tr>
<td>1</td>
<td>YES</td>
<td>A batch operation is under way.</td>
</tr>
</tbody>
</table>

Example

// check if a batch operation is active
if (CTRecord::IsBatchActive())
   printf("Batch operation underway\n");
else
   printf("No batch operation\n");

See Also

CTRecord::BatchLoaded(), CTRecord::BatchLocked(), CTRecord::BatchMode(),
CTRecord::BatchTotal(), CTRecord::EndBatch(), CTRecord::InsertBatch(),
CTRecord::NextBatch(), CTRecord::SetBatch()
CTRecord::IsEdited

Syntax

CTBOOL IsEdited( ) const

Parameters

This method has no parameters.

Description

Indicates if a record buffer has been modified.

Return

IsEdited() returns YES if the record has been modified, NO otherwise

See also

Write(), SetEdited()
CTRecord::IsFiltered

Indicates if a filter condition is active for this record object.

Declaration
CTBOOL CTRecord::IsFiltered();

Description
IsFiltered indicates whether a filter condition is active for this record object.

Return

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO</td>
<td>No filter condition is active.</td>
</tr>
<tr>
<td>1</td>
<td>YES</td>
<td>A filter condition is active.</td>
</tr>
</tbody>
</table>

Example
// if the record filter is active, retrieve the filter expression
if  (hRecord.IsFiltered())
{
    CTString expr;
    hRecord.GetFilter(expr);
    printf("Filter expression %s\n", expr.c_str());
}

See Also
CTRecord::SetFilter(), CTRecord::GetFilter()
CTRecord::isNew

Syntax

CTBOOL IsNew( ) const

Parameters

This method has no parameters.

Description

Indicates if a record buffer is cleared.

Return

IsNew() returns YES if the record buffer is cleared, NO otherwise.

See also

Clear(), SetNew()
CTRecord::IsNullField

Syntax
CTBOOL IsNullField(NINT FieldNumber)
CTBOOL IsNullField(const CTString& FieldName)

Parameters
- FieldNumber [in] The field number.
- FieldName [in] The field name.

Description
Indicates if the field specified by field number or name is a null field.

Return
IsNullField() returns YES if the field is null, NO otherwise.
**CTRecord::IsRangeOn**

Indicate if an index range operation is active for this record handle.

**Declaration**

```cpp
bool CTRecord::IsRangeOn()
```

**Description**

`CTRecord::IsRangeOn()` returns YES if an index range operation is active for this record handle, or NO if no index range is active.

**Return**

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO</td>
<td>No index range is active.</td>
</tr>
<tr>
<td>1</td>
<td>YES</td>
<td>An index range is active.</td>
</tr>
</tbody>
</table>

**Example**

```cpp
if (pRec->IsRangeOn())
    printf("Range is on\n");
```

**See Also**

`CTRecord::RangeOn()`, `CTRecord::RangeOff()`
CTRecord::IsRecordSetOn

Declaration
CTBOOL CTRecord::IsRecordSetOn();

Description
Indicates if a record set is active or not. A record set is active after a successful call to RecordSetOn(). A record set can be switched off by calling RecordSetOff().

Return Values
Returns YES if a record set if active and NO if a record set is not active.

See Also
CTRecord::RecordSetOn(), CTRecord::RecordSetOff()
CTRecord::IsVariableField

Syntax
CTBOOL IsVariableField(NINT FieldNbr)
CTBOOL IsVariableField(const CTString& FieldName)

Parameters
- FieldNbr [in] The field number.
- FieldName [in] The field name.

Description
Indicates if a field is in the variable portion of a record

Return
IsVariableField() returns YES if a field is in the variable portion of a record, and NO otherwise.
CTRecord::Last

Syntax
CTBOOL Last ( )

Parameters
This method has no parameters.

Description
Moves to the last record in the table.

Return
Last() returns YES if the record is found, NO otherwise.

See also
First(), Next(), Prev(), Find()
CTRecord::LockRecord

Syntax
void LockRecord(CTLOCK_MODE mode)

Parameters
- **mode** [in] the lock mode. Valid values for the lock mode are shown in the "c-treeDB definitions" (page 227).

Description
Locks the current record. In order to use this method, a record must be considered current. A record is set as the current record by update methods (**Write()** or search methods (**Find()**, **FindTarget()**, **First()**, **Next()**, **Prev()**, **Last()**, **SeekRecord()**).

After a Record has been cleared (**Clear()**), no current record buffer is kept, and **LockRecord()** cannot be used. In this case, use the **CTBase** Lock method.

**UnlockRecord()** may be used to free the record lock. **LockRecord(CTLOCK_FREE)** has the same effect. **Unlock()** and **Lock(CTLOCK_FREE)** free all session wide locks but DO NOT free any records locked with **LockRecord()**.

A record that is locked with one of the READ locks available allows any other user with a READ lock to read that record. No one can update one record using a READ lock. When one user gets a WRITE lock in a record, it means he/she may update that particular record, and until the record is freed, no one else is able to read that particular record.

Notice that **LockRecord()** should be called BEFORE modifying the record contents; otherwise, the changes will be lost.

When working with **LockRecord()**, the user must be certain to release the lock before moving to the next record or will need to use **CTTable::UnlockTable()** to release all record locks from a table at once. Do not mix the use of **CTBase::Lock()** and **LockRecord()** since an error DLOK_ERR, indicating record already locked, may occur.

Return
None.

See also
**UnlockRecord()**, **CTBase::Lock()**
CTRecord::Next

Syntax

CTBOOL Next( )

Parameters

This method has no parameters.

Description

Moves to the next record in the table. Before calling Next for the first time in a table, one of the Record search methods must be used: Find() or First().

Return

Next() returns YES if the record is found, NO otherwise.

Example

pRec->First();
do {
    printf("\nName: %s", pRec->GetFieldAsString("name").c_str());
} while (pRec->Next());

See also

First(), Prev(), Last(), Find()
CTRecord::NextBatch

Retrieves the next record from the batch buffer.

Declaration

```cpp
void CTRecord::NextBatch()
```

Description

If the mode of the batch operation is `CTBATCH_GET`, `CTBATCH_RANGE`, or `CTBATCH_PHYS` then it may be necessary to retrieve all records that match the batch criteria. The records are retrieved by calling `CTRecord::NextBatch()` method.

`CTRecord::NextBatch()` method retrieves a record data from the batch buffer maintained by c-treeDB’s record object. After a successful call to `CTRecord::NextBatch()` method, the field data can be retrieved by calling the appropriate `CTRecord::GetFieldAs...()` methods.

In case of error, `CTRecord::NextBatch()` throws an exception.

Return

The user will receive an exception if the batch has ended.

Example

```cpp
CTRecord hRecord;
    while (true)
    {
        try
        {
            hRecord.NextBatch();
        }
        catch (CTException err)
        {
            if (err.GetErrorCode() == INOT_ERR)
                break;
            else
                throw err;
        }
    CTString invoice = hRecord.GetFieldAsString("Invoice");
    CTString item = hRecord.GetFieldAsString("Item");
    printf("%-11s %s\n", invoice.c_str(), item.c_str());
    }
```

See Also

`CTRecord::BatchLoaded()`, `CTRecord::BatchLocked()`, `CTRecord::BatchMode()`, `CTRecord::BatchTotal()`, `CTRecord::EndBatch()`, `CTRecord::InsertBatch()`, `CTRecord::IsBatchActive()`, `CTRecord::SetBatch()`
CTRecord::NextInBatch

Retrieves the next batch of records from batch retrieval operation.

Declaration

```cpp
bool CTRecord::NextInBatch();
```

Description

If the mode of the batch operation is one of `CTBATCH_GET`, `CTBATCH_RANGE` or `CTBATCH_PHYS` then it may be necessary to retrieve all records that match the batch criteria. The records are retrieved by calling `NextInBatch()` method. `NextInBatch()` method retrieve the record data from the batch buffer maintained by c-treeDB's record handle. After a successful call to `NextInBatch()` function the field data can be retrieved by calling the appropriate `GetFieldAs...()` methods.

Return

YES if successful, NO if no more record match the criteria.

Throws a `CTException` on error.

`NextInBatch()` does not throw an exception when the end of the batch is reached.

See Also

`BatchLocked()`, `BatchMode()`, `EndBatch()`, `InsertBatch()`, `IsBatchActive()`, `NextBatch()`, `SetBatch()`, `BatchLoaded`
CTRecord::NumberOfKeyEntries

Syntax

```cpp
LONG CTRecord::NumberOfKeyEntries(const CTString& indexName);
```

Parameters

`indexName` The name of the index to read the number of entries from.

Description

`NumberOfKeyEntries()` retrieves the number key entries in an index file identified by `indexName`.

Return

`NumberOfKeyEntries()` returns the number of key entries in the index. If an error is detected, `NumberOfKeyEntries()` throws a `CTException` exception.

See also

`CTRecord::EstimateSpan()`
CTRecord::Prev

Syntax
CTBOOL Prev( )

Parameters
This method has no parameters.

Description
Moves to the previous record in the table. Before calling Prev() for the first time in a table, one of the Record search methods must be used: Find() or Last().

Return
Prev() returns YES if the record is found, NO otherwise.

See also
First(), Next(), Prev(), Find()
**CTRecord::RangeOff**

Terminates a record index range operation established by `CTRecord::RangeOff()`.

**Declaration**

```cpp
void CTRecord::RangeOff()
```

**Description**

`CTRecord::RangeOff()` terminates a range operation.

**Return**

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CTDBRET_OK</td>
<td><code>CTRecord::RangeOff()</code> returns CTDBRET_OK on success or c-treeDB SDK error code on failure.</td>
</tr>
</tbody>
</table>

See "c-treeDB Errors and Return Values" (page 881) for a complete listing of valid c-treeDB error codes and return values.

**Example**

```cpp
pRec->RangeOff();
```

**See also**

`CTRecord::RangeOn()`, `CTRecord::IsRangeOn()`
**CTRecord::RangeOn**

Establish a new index range on a record handle.

**Declaration**

```cpp
void CTRecord::RangeOn(NINT SegCount, pVOID lRange, pVOID uRange, pNINT operators)
```

**Description**

`CTRecord::RangeOn()` establishes a new range based on the key segment values passed on `lRange` and `uRange` buffers, and the operators for each segment. Once the range is set, use `CTRecord::First()`, `CTRecord::Next()`, `CTRecord::Prev()` and `CTRecord::Last()` to navigate the records in the specified range. The range is set for all index entries that are situated between the lower bounds and upper bounds values. The segment values are stored in `lRange` and `uRange` buffers in the same order and type of the index segment definition. If a previous range exists for this index, the previous range is released and the new range is established. Ranges take precedence over sets. If a record handle has a set established, record from a range will fetched instead of records from a range. Once the range is terminated, the records from a set is established.

- **SegCount** indicates the number of index segments values that should be used for setting the range, and the number of operators, since there must be one operator for each key segment in `lRange` and/or `uRange`.
- **lRange** is a buffer with the lower range segment values. Use the function `CTRecord::BuildTargetKey()` to build the `lRange` buffer.
- **uRange** is a buffer with the upper range segment values. Use the function `CTRecord::BuildTargetKey()` to build the `uRange` buffer.
- **operators** is an array of operators. There must be one operator for each key segment in `lRange` and/or `uRange`. The operators `CTIX_EQ`, `CTIX_NE`, `CTIX_GT`, `CTIX_GE`, `CTIX_LE`, `CTIX_LT` are open ended and use only the `lRange` buffer for range values and the equivalent key segment in `uRange` is ignored and maybe set to null (ascii \0 values). The operators `CTIX_BET`, `CTIX_BET_IE`, `CTIX_BET_EI`, `CTIX_BET_EE` and `CTIX_NOTBET` use both `lRange` and `uRange` buffers to establish the lower and upper bound values.

**Return**

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CTDBRET_OK</td>
<td><code>CTRecord::RangeOn()</code> returns CTDBRET_OK on success or c-treeDB SDK error code on failure.</td>
</tr>
</tbody>
</table>

See "c-treeDB Errors and Return Values" (page 881) for a complete listing of valid c-treeDB error codes and return values.

**See also**

`CTRecord::RangeOff()`, `CTRecord::IsRangeOn()`
CTRecord::Read

**Syntax**

```cpp
void Read();
```

**Parameters**

This method has no parameters.

**Description**

Reads the current record.

**Return**

None.

**See also**

Clear(), Write(), Delete(), Reset()
CTRecord::RecordSetOff

Syntax
void RecordSetOff( )

Parameters
This method has no parameters.

Description
Deactivates record sets.

Return
None.

Example
pRec->RecordSetOn(5);
pRec->SetDefaultIndex("last name");
pRec->SetFieldAsString("last name", "silva");
pRec->First();
pRec->RecordSetOff();

See also
RecordSetOn()
CTRecord::RecordSetOn

Syntax
void RecordSetOn(NINT siglen)

Parameters
- **siglen** [in] The number of key bytes.

Description
Activates record sets. After activating a record set, set the fields and index desired in the search, and then perform the search using the regular search functions. Notice that it is necessary to set the first **siglen** bytes from the index segments.

Return
None.

Example

```c
// display all records in set - no error checking
void DisplayAll(CTRecord& pRec)
{
    NINT count = 0;
    pRec.Clear();
    pRec.SetDefaultIndex("index_name");
    pRec.SetFieldAsString(0, "silva");
    pRec.RecordSetOn(5);
    if (pRec.First())
    {
        do
        {
            count++;
            PrintRecord(pRec);
        }
        while (pRec.Next());
    }
    printf("%d records in set\n", count);
}
```

See also
RecordSetOff()
CTRecord::Reset

Syntax
void Reset();

Parameters
This method has no parameters.

Description
Resets the record buffer to its initial condition.

Return
None.

See also
Clear(), Read(), Write(), Delete()
CTRecord::SeekRecord

Syntax
void SeekRecord(CTOFFSET offset)

Parameters

Description
Moves the record to the position pointed to by offset.

Return
None.

See also
GetRecordPos(), SetRecordPos(), SetRecordOffset()
CTRecord::SetBatch

Perform operations on a group of records.

Declaration

void CTRecord::SetBatch(CTBATCH_MODE mode, VRLEN targetLen, VRLEN bufferLen);

Description

CTRecord::SetBatch() attempts to initiate a specified operation on a group of records with keys matching a partial key value, an index range expression, or the entire table by physical order.

The mode parameter specifies which batch operation is to take place. You must choose at least one of the mandatory modes. You may choose one or more of the optional modes to specify further parameters for the batch operation. Please refer to the description of the modes below.

- **targetLen** - the number of significant bytes of the partial target key when the batch mode is `CTBATCH_GET` or `CTBATCH_DEL`.
- **bufferLen** - the size of the buffer used internally by c-treeDB code to handle batch operations. A zero value for this parameter is an indication that the default buffer size should be used. The default buffer size is calculated as the size of the fixed portion of the record multiplied by 128.

In case of errors, **CTRecord::NextBatch()** throws a **CTException**.

Mandatory modes

<table>
<thead>
<tr>
<th>MODE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CTBATCH_GET</code></td>
<td>Retrieve a group of related records by partial key</td>
</tr>
<tr>
<td><code>CTBATCH_RANGE</code></td>
<td>Retrieve a group of related records based on an index range expression</td>
</tr>
<tr>
<td><code>CTBATCH_PHYS</code></td>
<td>Retrieve records from a table in physical order. The starting record for the batch retrieval may be specified.</td>
</tr>
<tr>
<td><code>CTBATCH_DEL</code></td>
<td>Delete a group of related records by partial key</td>
</tr>
<tr>
<td><code>CTBATCH_INS</code></td>
<td>Insert a group of records</td>
</tr>
</tbody>
</table>

Optional modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CTBATCH_GKEY</code></td>
<td>Process records with a greater than or equal key match with the target key. When this mode is specified, the number of matched records is not readily available. <code>ctdbBatchLocked()</code> and <code>CTRecord::BatchLocked()</code> returns a value one greater than <code>ctdbBatchLoaded()</code> to indicate there may be more records to process. This mode is applicable only with <code>CTBATCH_GET</code> and <code>CTBATCH_DEL</code> modes and can not be used with <code>CTBATCH_LKEY</code>.</td>
</tr>
<tr>
<td><code>CTBATCH_LKEY</code></td>
<td>Process records that have a less than or equal key match with the target key. This mode is applicable only with <code>CTBATCH_GET</code> and <code>CTBATCH_DEL</code> modes and can not be used with <code>CTBATCH_GKEY</code>.</td>
</tr>
<tr>
<td><code>CTBATCH_VERIFY</code></td>
<td>Verify that the keys in the index match the values in the key fields of the</td>
</tr>
</tbody>
</table>
## Mode Description

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>record.</td>
</tr>
<tr>
<td><strong>CTBATCH_LOCK_KEEP</strong></td>
<td>Keep all records locked after ...EndBatch() is called. Without this mode, all records locks are released when ...EndBatch() is called. This option is only in effect when used with CTBATCH_LOCK_READ or CTBATCH_LOCK_WRITE.</td>
</tr>
<tr>
<td><strong>CTBATCH_LOCK_READ</strong></td>
<td>Place a read lock on each record that matches the partial key.</td>
</tr>
<tr>
<td><strong>CTBATCH_LOCK_WRITE</strong></td>
<td>Place a write lock on each record that matches the partial key.</td>
</tr>
<tr>
<td><strong>CTBATCH_LOCK_BLOCK</strong></td>
<td>Convert a CTBATCH_LOCK_READ or CTBATCH_LOCK_WRITE to blocking read and blocking write locks, respectively.</td>
</tr>
<tr>
<td><strong>CTBATCH_LOCK_ONE</strong></td>
<td>Implement an alternative locking strategy: only locks the record during the record read; original locking strategy keeps locks on during entire batch processing.</td>
</tr>
<tr>
<td><strong>CTBATCH_COMPLETE</strong></td>
<td>...SetBatch() returns a success code only if all matching records are successfully locked. You must specify either CTBATCH_LOCK_READ or CTBATCH_LOCK_WRITE.</td>
</tr>
</tbody>
</table>

### Retrieving records by partial key

All records with key matching a partial target key are loaded into a buffer region maintained internally by c-treeDB. If the selected records do not fit in the buffer, those that fit are loaded, and subsequent calls will retrieve the remaining records.

The following steps must be taken to perform a retrieval batch operation based on a partial key:

1. Clear a record buffer by calling the CTRecord::Clear() method.
2. Use the CTRecord::SetFieldAs...() methods to set the fields that form the partial target key that will be used to select a group of records.
3. Call the CTRecord::SetBatch() method, with CTBATCH_GET mode, to start a new record retrieval batch operation.
4. If the CTRecord::SetBatch() method returns with no errors, call the CTRecord::NextBatch() method repeatedly until all related records are retrieved. CTRecord::NextBatch() returns BTMT_ERR (428) to indicate no more records are available.

When you are done with the batch records, call the CTRecord::EndBatch() method to terminate the batch operation. Please note that another batch operation can only start after the current batch operation is terminated.

### Retrieving records by index range

All records that match an index range expression are loaded into a buffer region maintained internally by c-treeDB. If the selected records do not fit in the buffer, those that fit are loaded, and subsequent calls will retrieve the remaining records.

The following steps must be taken to perform an index range batch retrieval of records:

1. Establish an index range by calling CTRecord::RangeOn() method;
2. Call the CTRecord::SetBatch() method with the CTBATCH_RANGE mode to start a new record retrieval batch operation.
3. If the CTRecord::SetBatch() method returns with no errors, call the CTRecord::NextBatch() method repeatedly until all related records are retrieved. CTRecord::NextBatch() returns BTMT_ERR (428) to indicate no more records are available.

4. When you are done with the batch records, call the CTRecord::EndBatch() method to terminate the batch operation.

5. Call the CTRecord::RangeOff() method to terminate index range operation.

Retrieving records by physical order

All records of a table are loaded by physical order into a buffer region maintained internally by c-treeDB. If the selected records do not fit in the buffer, those that fit are loaded, and subsequent calls will retrieve the remaining records.

The following steps must be taken to perform a physical order batch retrieval of records:

1. Call the CTRecord::SetBatch() method with the CTBATCH_PHYS mode to start a new record retrieval batch operation.

2. If the CTRecord::SetBatch() method returns with no errors, call the CTRecord::NextBatch() method repeatedly until all related records are retrieved. CTRecord::NextBatch() returns BTMT_ERR (428) to indicate no more records are available.

3. When you are done with the batch records, call the CTRecord::EndBatch() method to terminate the batch operation.

**Note:** Setting a batch with CTBATCH_PHYS will cause slightly different behavior from setting it with CTBATCH_GET.

If the number of records exceeds the size of the buffer set when calling SetBatch, the total returned by BatchTotal will be only the number of records that fit into the batch buffer for CTBATCH_PHYS batches. If the batch was set with the CTBATCH_GET mode, the total number of records satisfying the batch will be returned, regardless if they all fit in the batch buffer. If a precise count of the number of records in a file is necessary, use GetRecordCount when you are in CTBATCH_PHYS mode.

This difference also affects record locking. If the batch was set with CTBATCH_PHYS, the records are locked when they are read into the buffer, so only the records that have been read into the batch buffer are locked. If the batch was set with CTBATCH_GET, all records are locked on the initial call.

Deleting a group of records

If the intended batch operation is to delete a group of selected records, you need to initially set the partial target key to select the group of related records and then start the batch operation to delete the selected records.

Even if no records are retrieved with the delete operation, CTRecord::EndBatch() must be called to terminate the current batch operation.

The following steps must be taken to perform a batch delete record operation:

1. Clear a record buffer by calling the CTRecord::Clear() method.

2. Use the CTRecord::SetFieldAs...() methods to set the fields that form the partial target key that will be used to select a group of records.

3. Call the CTRecord::SetBatch() methods with CTBATCH_DEL mode to delete a group of related records.

4. Call the CTRecord::EndBatch() method to terminate the delete record batch operation.
Inserting a group of records

A group of new records are loaded into a buffer region maintained internally by c-treeDB and this group of records are inserted into a table.

When the batch buffer fills up, the group of records stored in the batch buffer are inserted into the table. If `CTRecord::EndBatch()` is called and the batch buffer still contains records, a new insert record operation is performed for the remaining records before the batch operation is terminated.

For transaction controlled files, the batch insertion operation is treated as one all or nothing operation. If no explicit transaction is started, each insertion of records will start and end its own transaction. Even if an explicit transaction is started, each insertion operation is treated independently through safe points.

**Note:** currently, all records insertion operations will not perform any conversion of records images, key values and records position for heterogeneous client/server implementations.

The following steps must be taken to perform a batch insert record operation:

1. Call the `CTRecord::SetBatch()` method with `CTBATCH_INS` mode to insert a group of records.
   
   For each record to be inserted perform the following operations:
   
   a. call `CTRecord::Clear()` to initialize a record buffer
   
   b. for each field in the record call one of the `CTRecord::SetFieldAs...()` method to set the field data;
   
   c. call `CTRecord::InsertBatch()` to insert the record into the batch buffer

2. Call `CTRecord::EndBatch()` to indicate that no more records will be inserted.

**Return**

tvoid

**Example**

```c
void GetInvoiceItems(CTRecord& hRecord, NINT Invoice)
{
    NINT count = 0;

    try
    {
        // set the partial target key */
        hRecord.Clear();
        hRecord.SetFieldAsSigned("Invoice", Invoice);
        // set the batch operation
        hRecord.SetBatch(CTBATCH_GET, sizeof(Invoice), 0);

        // retrieve records
        while (hRecord.NextBatch())
            count++;

        // terminate batch operations
        hRecord.EndBatch();

        printf("%d records found\n", count);
    }
    catch (CTException& err)
    {
        printf("Batch failed [error %d]\n", hRecord.GetError());
    }
}
```
See Also

CTRecord::BatchLoaded(), CTRecord::BatchLocked(), CTRecord::BatchMode(),
CTRecord::BatchTotal(), CTRecord::EndBatch(), CTRecord::InsertBatch(),
CTRecord::IsBatchActive(), CTRecord::NextBatch()
**CTRecord::SetDefaultIndex**

**Syntax**
void SetDefaultIndex(NINT indexno)
void SetDefaultIndex(const CTString& name)

**Parameters**
- *name* [in] The new default record index name.

**Description**
Sets the new record default index. Unless this function is called, the first index is the default. To force a physical data table traversal without using any indices, specify the following constant:

- *CTDB_DATA_IDXNO*

**Return**
None.

**Example**

```cpp
myRecord->SetDefaultIndex("name");
// change the default index to "name"
myRecord->First();
// find the first record ordered by "name"
```

**See also**
GetDefaultIndex()
CTRecord::SetEdited

Syntax
void SetEdited(CTBOOL flag)

Parameters

Description
Sets the changed record flag.

Return
None.

See also
IsEdited()
CTRecord::SetFieldAsBigint

**Syntax**

```cpp
void SetFieldAsBigint(NINT FieldNumber, const CTBigint& value)
void SetFieldAsBigint(const CTString& FieldName, CTBigint& value)
```

**Parameters**

- `FieldNumber` [in] The field number.
- `value` [in] The big integer object
- `FieldName` [in] The field name.

**Description**

Sets the field contents as a big integer object (64-bit).

**Return**

None.
CTRecord::SetFieldAsBlob

Syntax
void SetFieldAsBlob (NINT FieldNumber, CTBlob& value)
void SetFieldAsBlob(const CTString& FieldName, CTBlob& value)

Parameters
- FieldNumber [in] The field number.
- value [in] The blob object
- FieldName [in] The field name.

Description
Sets the field contents as a blob object.

Return
None.
CTRecord::SetFieldAsBool

Syntax
void SetFieldAsBool(NINT FieldNumber, CTBOOL value)
void SetFieldAsBool(const CString& FieldName, CTBOOL value)

Parameters
- FieldNumber [in] The field number.
- value [in] The boolean value
- FieldName [in] The field name.

Description
Sets the field contents as a boolean value

Return
None.
**CTRecord::SetFieldAsByte**

**Syntax**

```cpp
void SetFieldAsByte (NINT FieldNumber, UTEXT value)
void SetFieldAsByte(const CTString& FieldName, UTEXT value)
```

**Parameters**

- *FieldNumber* [in] The field number.
- *value* [in] The byte value
- *FieldName* [in] The field name.

**Description**

Sets the field contents as a byte value (unsigned one-byte integer)

**Return**

None.
CTRecord::SetFieldAsChar

Syntax
void SetFieldAsChar (NINT FieldNumber, TEXT value)
void SetFieldAsChar(const CTString& FieldName, TEXT value)

Parameters
- **FieldNumber** [in] The field number.
- **value** [in] The byte value
- **FieldName** [in] The field name.

Description
Sets the field contents as a TEXT value (signed one-byte integer).

Return
None.
CTRecord::SetFieldAsCurrency

Syntax
void SetFieldAsCurrency(NINT FieldNumber, const CTCurrency& value)
void SetFieldAsCurrency(const CTString& FieldName, const CTCurrency& value)

Parameters
- FieldNumber [in] The field number.
- value [in] The byte value
- FieldName [in] The field name.

Description
Sets the field contents as a CTCurrency (64-bit) object.

Return
None.
**CTRecord::SetFieldAsDate**

**Syntax**

```cpp
tvoid SetFieldAsDate (NINT FieldNumber, const CTDate& value)
tvoid SetFieldAsDate(const CTString& FieldName, const CTDate& value)
```

**Parameters**

- *FieldNumber* [in] The field number.
- *value* [in] The byte value
- *FieldName* [in] The field name.

**Description**

Sets the field contents as a date object.

**Return**

None.
### CTRecord::SetFieldAsDateTime

**Syntax**

```cpp
void SetFieldAsDateTime (NINT FieldNumber, const CTDateTime& value)
void SetFieldAsDateTime(const CTString& FieldName, const CTDateTime& value)
```

**Parameters**

- **FieldNumber** [in] The field number.
- **value** [in] The byte value
- **FieldName** [in] The field name.

**Description**

Sets the field contents as a date and time object.

**Return**

None.
CTRecord::SetFieldAsFloat

Syntax
void SetFieldAsFloat (NINT FieldNumber, CFLOAT value)
void SetFieldAsFloat(const CTString& FieldName, CFLOAT value)

Parameters
- **FieldNumber** [in] The field number.
- **value** [in] The byte value
- **FieldName** [in] The field name.

Description
Sets the field contents as a double value

Return
None.
CTRecord::SetFieldAsMoney

Syntax
void SetFieldAsMoney (NINT FieldNumber, const CTMoney& value)
void SetFieldAsMoney(const CTString& FieldName, const CTMoney& value)

Parameters
- FieldNumber [in] The field number.
- value [in] The byte value
- FieldName [in] The field name.

Description
Sets the field contents as a money object.

Return
None.
**CTRecord::SetFieldAsNumber**

**Syntax**

```c
void SetFieldAsNumber(NINT FieldNumber, const CTNumber& value)
void SetFieldAsNumber (const CTString& FieldName, const CTNumber& value)
```

**Parameters**

- **FieldNumber** [in] The field number.
- **value** [in] The byte value
- **FieldName** [in] The field name.

**Description**

Sets the field contents as a `CTNumber` object.

**Return**

None.
CTRecord::SetFieldAsShort

Syntax
void SetFieldAsShort (NINT FieldNumber, COUNT value)
void SetFieldAsShort(const CTString& FieldName, COUNT value)

Parameters
- **FieldNumber** [in] The field number.
- **value** [in] The byte value
- **FieldName** [in] The field name.

Description
Sets the field contents as a 2-bytes signed integer (COUNT).

Return
None.
**CTRecord::SetFieldAsSigned**

**Syntax**

void SetFieldAsSigned (NINT FieldNumber, CTSigned value)
void SetFieldAsSigned(const CTString& FieldName, CTSIGNED value)

**Parameters**

- **FieldNumber** [in] The field number.
- **value** [in] The byte value
- **FieldName** [in] The field name.

**Description**

Sets the field contents as an integer signed value

**Return**

None.
**CTRecord::SetFieldAsString**

**Syntax**

```cpp
void SetFieldAsString (NINT FieldNumber, const CTString& value)
void SetFieldAsString(const CTString& FieldName, const CTString& value)
```

**Parameters**

- **FieldName** [in] The field name.
- **value** [in] The byte value
- **FieldName** [in] The field name.

**Description**

Sets the field contents as a string object.

**Return**

None.
CTRecord::SetFieldAsTime

Syntax
void SetFieldAsTime (NINT FieldNumber, const CTtime& value)
void SetFieldAsTime(const CString& FieldName, const CTtime& value)

Parameters
- FieldNumber [in] The field number.
- value [in] The time object
- FieldName [in] The field name.

Description
Sets the field contents as a time object.

Return
None.
CTRecord::SetFieldAsUnsigned

Syntax
void SetFieldAsUnsigned (NINT FieldNumber, CTUNSIGNED value)
void SetFieldAsUnsigned(const CTString& FieldName, CTUNSIGNED value)

Parameters
- **FieldName** [in] The field name.
- **value** [in] The unsigned integer value
- **FieldName** [in] The field name.

Description
Sets the field contents as an integer unsigned value

Return
None.
**CTRecord::SetFieldAsUTF16**

Sets the field with a Unicode UTF-16 string.

**Declaration**

```cpp
void CTRecord::SetFieldAsUTF16(NINT FieldNumber, pWCHAR value);
```

**Description**

- **FieldNumber** is a number representing the field number.
- **value** is the wide (UTF-16) string buffer.

**SetFieldAsUTF16()** sets the field with a UNICODE UTF-16 string. If the underlying field type is not one of the UNICODE field types, the UTF-16 string is converted to the appropriate type before the data is stores in the field.

**Return**

void

**Example**

```cpp
void AddData(CTRecord& hRecord, const CTString& str, NINT val)
{
    WCHAR WStr[32];

    hRecord.Clear();
    ctdb_u8TOu16(str, WStr, sizeof(WStr));
    hRecord.SetFieldAsUTF16(0, WStr);
    hRecord.SetFieldAsSigned(1, (CTSIGNED)val);
    hRecord.Write();
}
```
CTRecord::SetFieldAsWord

Syntax
void SetFieldAsWord (NINT FieldNumber, UCOUNT value)
void SetFieldAsWord(const CTString& FieldName, UCOUNT value)

Parameters
- `FieldName [in]` The field name.
- `value [in]` The UCOUNT value
- `FieldNumber [in]` The field number.

Description
Sets the field contents as an unsigned two-byte integer (UCOUNT) value.

Return
None.
CTRecord::SetFilter

Sets or clears a record filter condition.

Declaration

```cpp
void CTRecord::SetFilter(const CTString& cndexpr);
```

Description

- **cndexpr** [in] the filtering expression. The valid expressions are shown in the table below.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int atoi(char* String)</code></td>
<td>ASCII to integer.</td>
</tr>
<tr>
<td><code>int atol(char* String)</code></td>
<td>ASCII to long.</td>
</tr>
<tr>
<td><code>double atof(char* String)</code></td>
<td>ASCII to float.</td>
</tr>
<tr>
<td><code>int cabs( into Value )</code></td>
<td>Calculate the absolute value of a complex number.</td>
</tr>
<tr>
<td><code>int labs( into Value )</code></td>
<td>Calculate the absolute value of a long integer.</td>
</tr>
<tr>
<td><code>double fabs(double Value)</code></td>
<td>Calculate the absolute value of a float.</td>
</tr>
<tr>
<td><code>double ceil(double Value)</code></td>
<td>Calculate the ceiling of a value.</td>
</tr>
<tr>
<td><code>double floor(double Value)</code></td>
<td>Calculate the floor of a value.</td>
</tr>
<tr>
<td><code>double fmod(double r, double t)</code></td>
<td>Calculate the floating-point remainder.</td>
</tr>
<tr>
<td><code>int strlen(char* String )</code></td>
<td>Get the length of a string.</td>
</tr>
<tr>
<td><code>int strcmp(char* s, char* t)</code></td>
<td>Compare strings.</td>
</tr>
<tr>
<td><code>int stricmp(char* s, char* t)</code></td>
<td>Compare strings without regard to case.</td>
</tr>
<tr>
<td><code>int strncmp(char* s, char* t, int length )</code></td>
<td>Compare characters of two strings.</td>
</tr>
<tr>
<td><code>int strnicmp(char* s, char* t, int length )</code></td>
<td>Compare characters of two strings without regard to case.</td>
</tr>
</tbody>
</table>

Description

CTRecord::SetFilter() is used to set the filtering for a table. When set, all records retrieved from the table will be filtered against the expression, and just those records that match this criteria will be returned. Notice that this feature will be temporary, and just the user who sets the filter will have its records filtered. The filter is turned off when the table is closed, or when CTRecord::SetFilter() is called with NULL in the parameter **cndexpr**. If a new expression is set to a table with a current filter, the old filter is replaced with the new one. Only one filter may be active per table per user at a given time.

When used in the client/server model, this feature has the potential to increase the performance because only the records matching the criteria will be returned, reducing the network traffic. If used in conjunction with sets (CTRecord::RecordSetOn()), it may behave as a simple query.

Return

void
Example

// scan all records from California
void ScannAllRecord(CTTable &hTable)
{
    CTRecord hRecord(hTable);

    hRecord.SetFilter("strcmp(state, \"CA\") == 0");
    if (hRecord.First())
    {
        do
        {
            DisplayRecord(hRecord);
        }
        while (hRecord.Next());
    }
}

See Also

CTRecord::GetFilter(). CTRecord::IsFiltered()
CTRecord::SetNew

Syntax
void SetNew(CTBOOL flag)

Parameters

Description
Sets the new record flag.

Return
None.

See also
IsNew()
CTRecord::SetRecordOffset

Syntax
void SetRecordOffset(CTOFFSET offset)

Parameters

Description
Updates the record offset. The current record pointer is not moved and no record data is updated.

Return
None.

See also
GetRecordPos(), SetRecordPos(), SeekRecord()
CTRecord::SetRecordPos

Syntax
void SetRecordPos(CTOFFSET offset)

Parameters

Description
Sets the current record offset position. The record buffers are not updated. In order to update the record buffer, Read must be called.

Return
None.

See also
GetRecordPos(), Read(), SetRecordOffset(), SeekRecord()
CTRecord::SwitchContext

Force a c-tree Plus ISAM context switch.

Declaration
void CTRecord::SwitchContext()

Description
Force a switch to the c-tree Plus ISAM context indicated by the record object. Each record object has its own c-tree ISAM context id.

When most c-treeDB record handling functions are called, they automatically perform a c-tree ISAM context switch. SwitchContext() may be useful before calling specific c-tree ISAM or low level calls to make sure the correct ISAM context is active before making those calls.

Return

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td></td>
<td>SwitchContext return no value.</td>
</tr>
</tbody>
</table>


EXAMPLE

// force a context switch
hRecord.SwitchContext();
// call ResetRecord
if (ResetRecord((COUNT)hTable.GetDatno(), SWTCURI))
    printf("ResetRecord failed\n");

See also
CTBase::SwitchInstance(), CTTable::GetDatno(), CTTable::GetIdxno(), CTIndex::GetIdxno()
**CTRecord::UnlockRecord**

**Syntax**

```cpp
void UnlockRecord( )
```

**Parameters**

None.

**Description**

Unlocks the current record. To free all record locks from one specific table obtained with `LockRecord()`, use `CTTable::UnlockTable()`.

**Return**

None.

**See also**

`LockRecord()`, `CTTable::UnlockTable()`
CTRecord::Write

Syntax
void Write();

Parameters
This method has no parameters.

Description
Writes the record buffer to disk. In multi-user environments, the record should be locked before an update.

Return
None.

See also
Clear(), Read(), Delete(), Reset(), LockRecord(), CTBase::Lock()
4.15 CTResource Class

class CTResource

Description
The \texttt{CTResource} class deals with c-tree resource concepts.

See also
\texttt{CTBase}

Preconditions
Resources must be defined, which is the default for c-treeDB tables.
CTResource Methods
CTResource::CTResource

CTResource constructor.

Declaration

CTResource(const CTTable& hTable);
CTResource(const CTTable& hTable, ULONG type, ULONG number);
CTResource(const CTTable& hTable, ULONG type, ULONG number, const CTString& name);

Description

CTResource constructor. The first constructor instantiate a resource object with the resource type and number set to zero and the resource name set to NULL. Set second overloaded constructor instantiate a resource object with a resource type and number, but the resource name set to NULL. The last overloaded constructor instantiate a resource object setting the resource type, number and name.

Return

None

Example

```c++
void DisplayAllResources(const CTTable& hTable)
{
    CTResource* hRes = new CTResource(hTable);
    try
    {
        // get the first resource
        if (hRes->First())
        {
            do
            {
                // display resource type, number and name
                printf("Resource type: %u, number: %u",
                        hRes->GetType(), hRes->GetNumber());
            } while (hRes->Next());
        }
    }
    catch (CTException &err)
    {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
        delete hRes;
    }
}
```

See Also

CTResource::Add(), CTResource::Delete(), CTResource::Update(), CTResource::First(), CTResource::Next(), CTResource::Find(), CTResource::GetType(), CTResource::SetType(), CTResource::GetNumber(), CTResource::SetNumber(), CTResource::GetName(), CTResource::SetName(), CTResource::GetDataLength(), CTResource::GetData(), CTResource::SetData(), CTResource::Unlock(), CTResource::IsLocked()
CTResource::Add

Add a new resource to a table.

Declaration

```c
void CTResource::Add(pVOID data, VRLEN size);
```

Description

The resource `data` is any collection of data that you wish to store as a Resource. It can be a character string, a structure, or any variable type. `size` indicates the number of bytes occupied by the data.

Adds a new resource to a table. When adding a Resource to a table, a special variable-length record is written to the table, containing the information from the Resource Data Block. This is done even if a data file uses fixed-length records. Every Resource is identified by a unique combination of a Resource Type and a Resource Number. The Resource Number can optionally be assigned by c-tree Plus during the call to `CTResource::Add()`. In addition, each Resource can be identified by a Resource Name. The Resource Name is not guaranteed to be unique.

The Resource Type must be a value greater than 65536. 0 through 65536 are reserved for FairCom use. If the Resource Number is a value of `CTDB_ASSIGN_RESOURCE_NUMBER` (0xffffffff), c-tree Plus assigns this Resource the next available Resource Number for this Resource Type in the specified data file. The assigned number can be retrieved by calling `ctdbGetResourceNumber` before the resource handle is released. The Resource Name is optional. Names starting with "FC!" or "RD!", are reserved for FairCom use.

Return

None

Example

```c
void AddMyResource(const CTTable& hTable, ULONG type, const CTString& name, pVOID data, VRLEN size)
{
    CTResource* hRes = new CTResource(hTable, type, CTDB_ASSIGN_RESOURCE_NUMBER, name);
    try
    {
        hRes->Add(data, size)
        printf("Resource added with number %u\n", hRes->GetNumber());
    }
    catch (CTException &err)
    {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
    }
    delete hRes;
}
```

See Also

`CTResource::Delete()`, `CTResource::Update()`, `CTResource::First()`, `CTResource::Next()`, `CTResource::Find()`, `CTResource::GetType()`, `CTResource::SetType()`, `CTResource::GetNumber()`, `CTResource::SetNumber()`, `CTResource::GetName()`.
CTResource::SetName(), CTResource::GetDataLength(), CTResource::GetData(), CTResource::SetData(), CTResource::Unlock(), CTResource::IsLocked()
CTResource::Delete

Delete a resource.

Declaration

void CTResource::Delete();

Description

Deletes a resource from a table. Before a resource can be deleted, the table must be opened exclusive. The resource type and resource number that identify this resource must be passed to one of the CTResource constructors.

Return

None

Example

void DelMyResource(const CTTable& hTable, ULONG type, ULONG number)
{
    CTResource* hRes = new CTResource(hTable, type, number);

    try
    {
        hRes->Delete();
    }
    catch (CTException &err)
    {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
    }
    delete hRes;
}

See Also

CTResource::Add(), CTResource::Update(), CTResource::First(), CTResource::Next(),
CTResource::Find(), CTResource::GetType(), CTResource::SetType(),
CTResource::GetNumber(), CTResource::SetNumber(), CTResource::GetName(),
CTResource::SetName(), CTResource::GetDataLength(), CTResource::GetData(),
CTResource::SetData(), CTResource::Unlock(), CTResource::IsLocked()
CTResource::Find

Find a resource by type and number of by name.

Declaration

CTBOOL CTResource::Find(ULONG type, ULONG number, CTBOOL lock) const;
CTBOOL CTResource::Find(const CTString& name, CTBOOL lock) const;

Description

The first overloaded method locates and retrieves a resource in a table based on type and number. Parameters type and number are values that should uniquely identify the resource and lock is used to indicate if the resource should be locked, if it is found.

The second overloaded method locates and retrieves a resource by name. c-tree Plus cannot guarantee unique resource names. Parameter name is the resource name and lock is used to indicate if the resource should be locked, if it is found.

Return

CTResource::Find() return YES if the resource was located and retrieved or NO if the resource could not be found. In case of error, a CException is thrown.

Example

// display a particular resource
void DisplayResource(const CTTable& hTable, ULONG type, ULONG number) {
    CTResource* hRes = new CTResource(hTable);
    try {
        if (hRes->Find(type, number)) {
            printf("Resource type: %u, number: %u, name: %s\n",
                   hRes->GetType(), hRes->GetNumber(),
                   hRes->GetName().c_str());
        } else {
            printf("Resource %d,%d not found\n", type, number);
        }
    } catch (CException &err) {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
    }
    delete hRes;
}

See Also

CTResource::Add(), CTResource::Delete(), CTResource::Update(), CTResource::First(),
CTResource::Next(), CTResource::GetType(), CTResource::SetType(),
CTResource::GetNumber(), CTResource::SetNumber(), CTResource::GetName(),
CTResource::SetName(), CTResource::GetDataLength(), CTResource::GetData(),
CTResource::SetData(), CTResource::Unlock(), CTResource::IsLocked()
CTResource::First

Retrieve the first resource stored in a table.

Declaration

CTBOOL CTResource::First(CTBOOL lock) const;

Description

Retrieves the first resource stored in a table. *lock* is used to indicate if the resource should be locked, if it is found.

Return

CTResource::First() returns YES if the first resource was retrieved or NO if first resource does not exist. In case of error, a *CTException* is thrown.

Example

```cpp
void DisplayAllResources(const CTTable& hTable)
{
    CTResource* hRes = new CTResource(hTable);

    try
    {  
        // get the first resource
        if (hRes->First())
        {
            do
            {
                // display resource type, number and name
                printf("Resource type: %u, number: %u", 
                       hRes->GetType(), hRes->GetNumber());
            } while (hRes->Next());
        }
    }
    catch (CTException &err)
    {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
        delete hRes;
    }
}
```

See Also

CTResource::Add(), CTResource::Delete(), CTResource::Update(), CTResource::Next(), CTResource::Find(), CTResource::GetType(), CTResource::SetType(), CTResource::GetNumber(), CTResource::SetNumber(), CTResource::GetName(), CTResource::SetName(), CTResource::GetDataLength(), CTResource::GetData(), CTResource::SetData(), CTResource::Unlock(), CTResource::IsLocked()
CTResource::GetData

Retrieve the resource data.

**Declaration**

pVOID CTResource::GetData() const;

**Description**

Retrieves the resource data

**Return**

Returns a pointer to the resource data.

**Example**

CTDBRET ReadMyResource(CTHANDLE Handle, ULONG type, ULONG number, pTEXT& data, VRLEN& size) {
    CTResource* hRes = new CTResource(Handle, type, number, NULL);
    try {
        if (hRes->Find(type, number))
        {
            size = hRes->GetDataLength();
            if (size > 0)
            {
                data = new TEXT[size];
                memcpy(data, hRes->GetData(), size);
            }
        }
    } catch (CTException &err) {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
        delete hRes;
    }
}

**See Also**

CTResource::Add(), CTResource::Delete(), CTResource::Update(), CTResource::First(),
CTResource::Next(), CTResource::Find(), CTResource::GetType(), CTResource::SetType(),
CTResource::GetNumber(), CTResource::SetNumber(), CTResource::GetName(),
CTResource::SetName(), CTResource::GetDataLength(), CTResource::SetData(),
CTResource::Unlock(), CTResource::IsLocked()
**CTResource::GetDataLength**

Retrieve the resource data length.

**Declaration**

```cpp
VRLEN CTResource::GetDataLength() const;
```

**Description**

Retrieves the resource data length in bytes.

**Return**

Returns the resource data length.

**Example**

```cpp
CTDBRET ReadMyResource(CTHANDLE Handle, ULONG type, ULONG number, pTEXT& data, VRLEN& size)
{
    CTResource* hRes = new CTResource(Handle, type, number, NULL);

    try
    {
        if (hRes->Find(type, number))
        {
            size = hRes->GetDataLength();
            if (size > 0)
            {
                data = new TEXT[size];
                memcpy(data, hRes->GetData(), size);
            }
        }
    }
    catch (CTException &err)
    {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
    }
    delete hRes;
}
```

**See Also**

CTResource::Add(), CTResource::Delete(), CTResource::Update(), CTResource::First(), CTResource::Next(), CTResource::Find(), CTResource::GetType(), CTResource::SetType(), CTResource::GetNumber(), CTResource::SetNumber(), CTResource::GetName(), CTResource::SetName(), CTResource::GetData(), CTResource::SetData(), CTResource::Unlock(), CTResource::IsLocked()
CTResource::GetName

Retrieve the resource name.

Declaration

CTString CTResource::GetName() const;

Description

Retrieves the resource name.

Return

Returns the resource name.

Example

if (hRes->GetName() != "MyResource")
  hRes->SetName("MyResource");

See Also

CTResource::Add(), CTResource::Delete(), CTResource::Update(), CTResource::First(),
CTResource::Next(), CTResource::Find(), CTResource::GetType(), CTResource::SetType(),
CTResource::GetNumber(), CTResource::SetNumber(), CTResource::SetName(),
CTResource::GetDataLength(), CTResource::GetData(), CTResource::SetData(),
CTResource::Unlock(), CTResource::IsLocked()
CTResource::GetNumber

Retrieve the resource number.

Declaration

ULONG CTResource::GetNumber() const;

Description

Retrieves the resource number.

Return

Returns the resource number.

Example

if (hRes->GetNumber() != number)
    hRes->SetNumber(number);

See Also

CTResource::Add(), CTResource::Delete(), CTResource::Update(), CTResource::First(),
CTResource::Next(), CTResource::Find(), CTResource::GetType(), CTResource::SetType(),
CTResource::SetNumber(), CTResource::GetName(), CTResource::SetName(),
CTResource::GetDataLength(), CTResource::GetData(), CTResource::SetData(),
CTResource::Unlock(), CTResource::IsLocked()
CTResource::GetType

Retrieve the resource type.

Declaration

LONG CTResource::GetType() const;

Description

Retrieves the resource type.

Return

Returns the resource type.

Example

if (hRes->GetType() != type)
    hRes->SetType(type);

See Also

CTResource::Add(), CTResource::Delete(), CTResource::Update(), CTResource::First(),
CTResource::Next(), CTResource::Find(), CTResource::SetType(),
CTResource::GetNumber(), CTResource::SetNumber(), CTResource::GetName(),
CTResource::SetName(), CTResource::GetDataLength(), CTResource::GetData(),
CTResource::SetData(), CTResource::Unlock(), CTResource::IsLocked()
CTResource::IsLocked
Check if resource is locked.

Declaration
CTBOOL CTResource::IsLocked() const;

Description
Check if a resource is locked.

Return
Returns YES is a resource is locked or NO is a resource is not locked.

Example
if (hRes->IsLocked())
  hRes->Unlock();

See Also
CTResource::Add(), CTResource::Delete(), CTResource::Update(), CTResource::First(),
CTResource::Next(), CTResource::Find(), CTResource::GetType(), CTResource::SetType(),
CTResource::GetNumber(), CTResource::SetNumber(), CTResource::GetName(),
CTResource::SetName(), CTResource::GetDataLength(), CTResource::GetData(),
CTResource::SetData(), CTResource::Unlock()
CTResource::Next

Retrieve the next resource stored in a table.

Declaration

CTBOOL CTResource::Next(CTBOOL lock) const;

Description

Retrieves the next resource stored in a table. lock is used to indicate if the resource should be locked, if it is found.

Return

CTResource::Next() returns YES if the first resource was retrieved or NO if first resource does not exist. In case of error, a CTEException is thrown.

Example

// read resources with type >= type and number > 0
void DisplayResources(const CTable& hTable, ULONG type)
{
    CTResource* hRes = new CTResource(hTable, type, 0);
    try
    {
        while (hRes->Next())
        {
            printf("Resource type: %u, number: %u",
                   hRes->GetType(), hRes->GetNumber());
        }
    }
    catch (CTException &err)
    {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
    }    
    delete hRes;
}

See Also

CTResource::Add(), CTResource::Delete(), CTResource::Update(), CTResource::First(), CTResource::Find(), CTResource::GetType(), CTResource::SetType(), CTResource::GetNumber(), CTResource::SetNumber(), CTResource::GetName(), CTResource::SetName(), CTResource::GetDataLength(), CTResource::GetData(), CTResource::SetData(), CTResource::Unlock(), CTResource::IsLocked()
CTResource::SetData

Set the resource data.

Declaration
void CTResource::SetData(pVOID data, VRLEN size);

Description
Sets the resource data.

Return
None

Example
CTDBRET ReadMyResource(CTHANDLE Handle, ULONG type, ULONG number, pTEXT& data, VRLEN& size) {
    CTResource* hRes = new CTResource(Handle, type, number, NULL);

    try {
        if (hRes->Find(type, number)) {
            size = hRes->GetDataLength();
            if (size > 0) {
                data = new TEXT[size];
                memcpy(data, hRes->GetData(), size);
            }
        }
    }
    catch (CTException &err) {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
        delete hRes;
    }
}

See Also
CTResource::Add(), CTResource::Delete(), CTResource::Update(), CTResource::First(),
CTResource::Next(), CTResource::Find(), CTResource::GetType(), CTResource::SetType(),
CTResource::GetNumber(), CTResource::SetNumber(), CTResource::GetName(),
CTResource::SetName(), CTResource::GetDataLength(), CTResource::GetData(),
CTResource::Unlock(), CTResource::IsLocked()
CTResource::SetName

Declaration

void CTResource::SetName(const CTString& name);

Description

Sets the resource name.

Return

None.

Example

if (hRes->GetName() != "MyResource")
    hRes->SetName("MyResource");

See Also

CTResource::Add(), CTResource::Delete(), CTResource::Update(), CTResource::First(), CTResource::Next(), CTResource::Find(), CTResource::GetType(), CTResource::SetType(), CTResource::GetNumber(), CTResource::SetNumber(), CTResource::GetName(), CTResource::GetDataLength(), CTResource::GetData(), CTResource::SetData(), CTResource::Unlock(), CTResource::IsLocked()
CTResource::SetNumber

Set the resource number.

**Declaration**

```cpp
void CTResource::SetNumber(ULONG number);
```

**Description**

Sets the resource number.

**Return**

None.

**Example**

```cpp
if (hRes->GetNumber() != number)
    hRes->SetNumber(number);
```

**See Also**

- `CTResource::Add()`, `CTResource::Delete()`, `CTResource::Update()`, `CTResource::First()`, `CTResource::Next()`, `CTResource::Find()`, `CTResource::GetType()`, `CTResource::SetType()`, `CTResource::GetNumber()`, `CTResource::GetName()`, `CTResource::SetName()`, `CTResource::GetDataLength()`, `CTResource::GetData()`, `CTResource::SetData()`, `CTResource::Unlock()`, `CTResource::IsLocked()`
CTResource::SetType

Set the resource type.

Declaration

```cpp
void CTResource::SetType(ULONG type);
```

Description

Sets the resource type.

Return

None

Example

```cpp
if (hRes->GetType() != type)
  hRes->SetType(type);
```

See Also

CTResource::Add(), CTResource::Delete(), CTResource::Update(), CTResource::First(),
CTResource::Next(), CTResource::Find(), CTResource::GetType(),
CTResource::GetNumber(), CTResource::SetNumber(), CTResource::GetName(),
CTResource::SetName(), CTResource::GetDataLength(), CTResource::GetData(),
CTResource::SetData(), CTResource::Unlock(), CTResource::IsLocked()
CTResource::Unlock

Unlock a resource.

Declaration

void CTResource::Unlock();

Description

Unlocks a resource. The resource is only unlocked if it was previously locked by First(), Next() or Find().

Return

None

Example

if (hRes->IsLocked())
    hRes->Unlock();

See Also

CTResource::Add(), CTResource::Delete(), CTResource::Update(), CTResource::First(), CTResource::Next(), CTResource::Find(), CTResource::GetType(), CTResource::SetType(), CTResource::GetNumber(), CTResource::SetNumber(), CTResource::GetName(), CTResource::SetName(), CTResource::GetDataLength(), CTResource::GetData(), CTResource::SetData(), CTResource::Unlock(), CTResource::IsLocked()
CTResource::Update

Update an existing resource.

Declaration

```cpp
void CTResource::Update(pVOID data, VRLEN size);
```

Description

Updates an existing resource. You must instantiate the resource object with the specific resource type and number that will uniquely identify the resource being updated. The resource `data` is any collection of data that you wish to store as a Resource. It can be a character string, a structure, or any variable type. `size` indicates the number of bytes occupied by `data`.

Return

None

Example

```cpp
void UpdateMyResource(const CTTable& hTable, ULONG type, ULONG number, pVOID data, VRLEN size)
{
    CTResource* hRes = new CTResource(hTable, type, number);

    try
    {
        hRes->Update(data, size);
    }
    catch (CTException &err)
    {
        printf("Error %d - %s\n", err.GetErrorCode(), err.GetErrorMsg());
    }
    delete hRes;
}
```

See Also

- `CTResource::Add()`, `CTResource::Delete()`, `CTResource::First()`, `CTResource::Next()`, `CTResource::Find()`, `CTResource::GetType()`, `CTResource::SetType()`, `CTResource::GetNumber()`, `CTResource::SetNumber()`, `CTResource::GetName()`, `CTResource::SetName()`, `CTResource::GetDataLength()`, `CTResource::GetData()`, `CTResource::SetData()`, `CTResource::Unlock()`, `CTResource::isLocked()`
4.16 CTSegment Class

class CTSegment

Description
Objects of the CTSegment class manage index segments. The segment is part of the index; one single index may be composed of several segments. It uses CTBase as the base class, and implements the constructor/destructor allocating/freeing memory to the segment operations.

See also
CTBase, CTable, CRecord, CField, CIndex

Preconditions
Before any task is developed with a CTSegment object, a CTIndex must have been initialized.

CTSegment Operators
- = : Assign a CTSegment object to another
CTSegment::operator=

**Syntax**

```cpp
CTSegment& operator=(const CTSegment& pSeg)
```

**Parameters**


**Description**

Assigns one `CTSegment` object to another

**Return**

Returns a `CTSegment` object
CTSegment Methods

Constructor / Destructor
- **CTSegment()**: Creates a *CTSegment* object
- **~CTSegment()**: Destroys a *CTSegment* Object and resets all the dependent objects

Segment Handling
- **GetField()**: Retrieves the segment field object.
- **GetFieldName()**: Retrieves the segment field name
- **GetMode()**: Retrieves the segment mode
- **SetMode()**: Sets the segment mode
- **GetNumber()**: Retrieves the segment index position in the index segments list
**CTSegment::CTSegment**

*Syntax*

```cpp
CTSegment ( )
CTSegment (const CTSegment& pSeg)
```

*Parameters*


*Description*

This is the constructor for the `CTSegment` Class.

*See also*

- `~CTSegment()`
CTSegment::~CTSegment

Syntax
~CTSegment

Parameters
This destructor has no parameters.

Description
This is the destructor for the CTSegment class.

See also
CTSegment
**CTSegment::GetField**

**Syntax**
```cpp
CTField GetField() const
```

**Parameters**
This method has no parameters.

**Description**
Retrieves the segment field object.

**Return**
`GetField()` returns the field object.

**See also**
`GetFieldName()`
CTSegment::GetFieldName

Syntax

CTString GetFieldName( ) const

Parameters

This method has no parameters.

Description

Retrieves the segment field name.

Return

GetFieldName() returns the segment field name.

See also

GetField()
**CTSegment::GetMode**

**Syntax**

```cpp
CTSEG_MODE GetMode( ) const
```

**Parameters**

This method has no parameters.

**Description**

Retrieves the segment mode. See `SetMode()` for valid values for the segment mode.

**Return**

`GetMode()` returns the segment mode.

**See also**

`SetMode()`
CTSegment::GetNumber

Syntax
NINT GetNumber( )

Parameters
This method has no parameters.

Description
Retrieves the segment index position in the index segments list

Return
GetNumber() returns the segment position.
CTSegment::GetPartialField

Retrieves the field object on which a key segment is based, even when the segment does not match the entire field length.

Declaration
CTField CTSegment::GetPartialField() const;

Description
c-treeDB was able to find the matching between a key segment and a field only if the key segment started at the beginning of a field and exactly matched the field length, as is required by SQL standards. It was desirable to retrieve a field on which the key segment start at the beginning of a field and does not exactly match the entire field.

The CTSegment::GetPartialField() method retrieves a field object that match a key, even when the segment does not match the entire field length.

Return
GetPartialField() returns a field object if a match can be found. If a match can't be found, GetPartialField() throws a CException object.

Example

```c
// find a field that match, or partial match, for the key segment
CTField GetFieldMatch(CTIndex &hIndex, NINT segnbr)
{
    return hIndex.GetSegment(segnbr).GetPartialField();
}
```

See Also
CTSegment::GetField(), CTSegment::GetFieldName(), CTSegment::GetPartialFieldName()
**CTSegment::GetPartialFieldName**

Retrieves the name of a field that match a key segment, even when the segment does not match the entire field length.

**Declaration**

```cpp
CTString CTSegment::GetPartialFieldName();
```

**Description**

c-treeDB was able to find the matching between a key segment and a field only if the key segment started at the beginning of a field and exactly matched the field length, as it is required by SQL. It was desirable to retrieve a field on which the key segment start at the beginning of a field and does not exactly match the entire field.

`CTSegment::GetPartialFieldName()` was added to retrieve the name of a field that match a key segment, even when the segment does not match the entire field length.

**Return**

`GetPartialFieldName()` returns a `CTString` object with the name of the field if a match can be found. If a match can't be found, `GetPartialFieldName()` throws a `CTException` object.

**Example**

```cpp
// Display the field name of a partial match
void PrintFieldMatch(CTIndex &hIndex, NINT segnbr)
{
    printf("Field name: %s\n",
            hIndex.GetSegment(segnbr).GetPartialFieldName().c_str());
}
```

**See Also**

`CTSegment::GetField()`, `CTSegment::GetFieldName()`, `CTSegment::GetPartialField()`
CTSegment::GetSegmentKSeg

Retrieves the current index wide extended key segment definition.

**Declaration**

```cpp
void CTSegment::GetSegmentKSeg(pctKSEGDEF pKSeg);
```

**Description**

`CTSegment::GetSegmentKSeg()` retrieves the current index wide extended key segment definition. `pKSeg` is a pointer to an extended key segment definition structure which will receive the definition.

**Return**

`void`

**Example**

```cpp
cTKSEGDEF kseg;

hSegment->GetSegmentKSeg(&kseg);
```

**See Also**

`CTIndex::GetIndexKSeg()`, `CTIndex::SetIndexKSeg()`, `CTSegment::SetKSegDefaults()`, `CTSegment::SetSegmentKSeg()`, `CTTable::GetTableKSeg()`, `CTTable::SetTableKSeg()`
**CTSegment::GetStatus**

Retrieves the status of the segment object.

**Declaration**

```cpp
ULONG CTSegment::GetStatus() const;
```

**Description**

Retrieves the status of a segment object. The status of the segment handle is a bit map describing one or more changes that have occurred to the segment object.

**Return**

`CTSegment::GetStatus()` returns a bitmap of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>CTDBISEG_OLD</td>
<td>Original segment as read from file</td>
</tr>
<tr>
<td>0x01</td>
<td>CTDBISEG_NEW</td>
<td>Segment added or inserted</td>
</tr>
<tr>
<td>0x02</td>
<td>CTDBISEG_DEL</td>
<td>Original segment deleted</td>
</tr>
<tr>
<td>0x04</td>
<td>CTDBISEG_MOVED</td>
<td>Original segment moved</td>
</tr>
<tr>
<td>0x10</td>
<td>CTDBISEG_MODE</td>
<td>Segment mode changed</td>
</tr>
<tr>
<td>0x20</td>
<td>CTDBISEG_FIELD</td>
<td>Segment field changed</td>
</tr>
</tbody>
</table>

**Example**

```cpp
// if any segment of an index has changed, call alter table
for (i = 0; i < hIndex->GetSegmentCount(); i++)
{
    CTSegment hSeg = hIndex->GetSegment(i);

    if (hSeg->GetStatus() != CTDBISEG_OLD)
        hTable.Alter(CTDB_ALTER_NORMAL);
}
```

**See Also**

`CTField::GetStatus()`, `CTIndex::GetStatus()`
CTSegment::MoveSegment

Moves a key segment.

Declaration

```c
void CTSegment::MoveSegment(NINT newIndex);
```

Description

`CTSegment::MoveSegment()` moves a key segment to a location indicated by `newIndex`. `newIndex` indicates the relative position where the key segment should be moved to.

Return

void

Example

```c
// move the last segment to first
CTIndex hIndex = hTable->GetIndex(0);
NINT count = hIndex->GetSegmentCount();

if (count > 0)
{
    CTSegment hSeg = hIndex->GetSegment(count - 1);
    hSeg->MoveSegment(0);
}
```

See Also

`CTTable::MoveSegment()`, `CTIndex::MoveSegment()`
CTSegment::SetKSegDefaults

Sets the system wide default values for the extended key segment definition.

Declaration

```c
void CTSegment::SetKSegDefaults(pctKSEGDEF pKSeg);
```

Description

Set the system wide default values for the extended key segment definition. The default values are:

- `kseg_ssiz` = ctKSEG_SSIZ_COMPUTED;
- `kseg_type` = ctKSEG_TYPE_UNICODE;
- `kseg_styp` = ctKSEG_STYP_UTF16;
- `kseg_comp` = ctKSEG_COMPU_S_DEFAULT | ctKSEG_COMPU_N_NONE;
- `kseg_desc` = "en_US"

Return

`void`

Example

```c
ctKSEGDEF kseg;
hSegment->SetKSegDefaults(&ksge);
```

See Also

CTIndex::GetIndexKSeg(), CTIndex::SetIndexKSeg(), CTSegment::GetSegmentKSeg(), CTSegment::SetSegmentKSeg(), CTTable::GetTableKSeg(), CTTable::SetTableKSeg()
CTSegment::SetMode

Syntax
void SetMode(CTSEG_MODE SegMode)

Parameters
- SegMode [in] The segment mode. Valid values for segment modes are shown in "Segment Modes" (page 92).

Description
Sets the segment mode.

Return
None.

See also
GetMode()
CTSegment::SetSegmentKSeg

Establishes a segment's extended key segment definition.

Declaration

```cpp
void CTSegment::SetSegmentKSeg(pctKSEGDEF pKSeg);
```

Description

`CTSegment::SetSegmentKSeg()` establishes a segment's extended key segment definition. `pKSeg` is a pointer to an extended key segment definition structure with the extended key definition.

Return

`void`

Example

```cpp
ctkKSEGDEF kseg;

kseg.kseg_ssiz = ctKSEG_SSIZ_COMPUTED;
kseg.kseg_type = ctKSEG_TYPE_UNICODE;
kseg.kseg_styp = ctKSEG_STYP_UTF16;
kseg.kseg_comp = ctKSEG_COMPU_S_DEFAULT | ctKSEG_COMPU_N_NONE;
kseg.kseg_desc = "en_US"

hSegment.SetSegmentKSeg(&kseg);
```

See Also

`CTIndex::GetIndexKSeg()`, `CTIndex::SetIndexKSeg()`, `CTSegment::GetSegmentKSeg()`, `CTSegment::SetKSegDefaults()`, `CTTable::GetTableKSeg()`, `CTTable::SetTableKSeg()`
4.17 CTSession Class

class CTSession

Description
Objects of the CTSession class manage sessions. A session is required to allow the connection with the databases; it is used for their general management as well. CTSession is derived from CTBase.

See also
CTBase, CTDatabase, CTTable

Preconditions
Since this is the upper level for the c-treeDB layer, it is the first required object to be created for any application.
**CTSession Methods**

**Constructor / Destructor**
- `CTSession()`: Creates a *CTSession* object
- `~CTSession()`: Destroys a *CTSession* Object and resets all the dependent objects

**Session Handling**
- `Create()`: Creates a new session dictionary
- `GetParam()`: Returns the session parameter based on the parameter type.
- `GetPassword()`: Returns the user password associated with the session.
- `GetPath()`: Retrieve the path (drive/directory) of the session dictionary.
- `GetPathPrefix()`: Get the client-side path prefix.
- `GetServerName()`: Returns the server name associated with the session.
- `GetUserLogonName()`: Returns the user name associated with the session.
- `IsActive()`: Checks if the session is active.
- `Logon()`: Logs on to c-tree Server, or c-tree instance, session.
- `Logout()`: Logs out from c-tree Server or from c-tree instance session and release all resources
- `SetParam()`: Sets the session parameter values.
- `SetPath()`: Sets the session dictionary path.
- `SetPathPrefix()`: Set the client-side path prefix.
- `GetLogonOnly()`: Retrieves the session logon only flag.
- `SetLogonOnly()`: Sets the session logon only flag. This flag, when set to YES before the session Logon, will prevent the session from using the session dictionary.
- `SetCurrentNodeName()`: Sets the client node name.

**DataBase Handling**
- `AddDatabase()`: Adds an existing database to the session.
- `CreateDatabase()`: Creates a new database
- `DeleteDatabase()`: Drops a database from the session and deletes the data and index files.
- `DisconnectAll()`: Disconnects all databases
- `DropDatabase()`: Drops a database from a session.
- `FindActive()`: Find active database by name or by uid
- `FindDatabase()`: Locates a database in a session.
- `FirstDatabase()`: Gets the first database for the session
- `GetDatabaseUID()`: Retrieves a database UID
- `GetFirstActive()`: Retrieves the first active database
- `GetNextActive()`: Retrieves the next active database
- `NextDatabase()`: Gets the Next database for the session
- `GetDatabaseCount()`: Retrieves the number of databases in the session dictionary.
CTSession::CTSession

Syntax
CTSession

Parameters
This constructor has no parameters.

Description
This is the constructor for the CTSession class.

See also
~CTSession()
CTSession::~CTSession

Syntax
~CTSession

Parameters
This destructor has no parameters.

Description
This is the destructor for the CTSession class. It checks to see if the session is still active, and closes it (logout) if so. It also frees the allocated memory of the Session.

See also
CTSession()
**CTSession::AddDatabase**

**Syntax**

```cpp
void AddDatabase (const CTString& Name, const CTString& Path)
```

**Parameters**

- **Name** [in] The database name to be added to the session
- **Path** [in] The database path

**Description**

Adds an existing database to the session.

**Return**

None.

**Example**

```cpp
pSession->Logon();
try
{
    pSession->AddDatabase(db_name, db_path);
    printf("\nDatabase added successfully!!");
}
catch (CTException& err)
{
    printf("\n\n%d  %s", err.GetErrorCode(),
                err.GetErrorMsg());
    return (1);
}
```

**See also**

`CreateDatabase()`, `DeleteDatabase()`, `DropDatabase()`
**CTSession::Attach**

Attaches an already initialized c-tree session to a c-treeDB session object.

**Declaration**

```cpp
void CTSession::Attach(CTBOOL isTransactionActive = NO);
void CTSession::Attach(const CTString& ctreeid, CTBOOL isTransactionActive = NO);
void CTSession::Attach(const CTSession& Session);
```

**Description**

- `isTransactionActive` indicates if a transaction is active or not.

**CTSession::Attach()** attaches an already initialized c-tree ISAM or low-level session to a c-treeDB session object. Attached sessions have no information on server, user name or user password, and these values are set to NULL. Parameter.

**Return**

`void`

**Example**

```cpp
// logon to c-tree using ISAM function
INTISAMX(10, 32, 32, 10, (USERPRF_NTKEY | USERPRF_CLRCHK));

// attach session to server handle
hSession->Attach(NO);

// ... do something useful ...

// detach from session
hSession->Detach();

// perform an ISAM logout
CLISAM();
```

**See Also**

- `CTSession::Detach()`
**CTSession::BeginImpersonation**

Enables the ability of the thread to support impersonation by another thread.

**Declaration**

```cpp
void CTSession::BeginImpersonation( NINT taskId )
```

**Description**

Start the impersonation of the specific task ID.

Stores the current "lock mode" in the `prevlockmode` session property and begins the impersonation for the given `taskId` (`ctImpersonateTask()`).

Retrieves the server side "lock mode" `ctdbGetLockMode()` after the impersonation and updates the `lockmode` session property.

**Note:** This function is not supported by the c-treeACE Server by default. It requires a custom build of the server available by special request.

**Return**

`void`

**See Also**

`CTSession::EndImpersonation()`
**CTSession::Create**

**Syntax**

```cpp
void Create (const CTString& dbEngine=ctdbDEFAULT_dbengine,
             const CTString& user=ctdbDEFAULT_userid,
             const CTString& password=ctdbDEFAULT_userword)
```

**Parameters**

- **dbEngine** [in] The c-tree Server Name
- **User** [in] The user name to use when logging on to the c-tree Server.
- **Password** [in] The user password to use when logging on to the c-tree Server.

**Description**

Creates a new session dictionary.

**Return**

None.

**See also**

Logon(), SetPath()
CTSession::CreateDatabase

Syntax
void CreateDatabase(const CTString& Name, const CTString& Path)

Parameters
- Name [in] the database name to be created
- Path [in] the database path

Description
Creates a new database.

Return
None.

See also
DeleteDatabase(), DropDatabase(), AddDatabase(), DisconnectAll()
CTSession::DeleteDatabase

Syntax
void DeleteDatabase(const CTString& Name)

Parameters
- Name [in] the database name to be deleted

Description
Drops a database from the session and deletes the data and index files.

Return
None.

See also
CreateDatabase(), DropDatabase(), AddDatabase()
CTSession::Detach

Detaches a c-treeDB session object from an existing c-tree connection.

Declaration

void CTSession::Detach();

Description

CTSession::Detach() detaches a c-treeDB session object from an existing c-tree connection. The c-tree ISAM or low-level un-initialization is not called, but the session object is re-initialized.

Return

void

Example

// logon to c-tree using ISAM function
INTISAMX(10, 32, 32, 10, (USERPRF_NTKEY | USERPRF_CLRCHK));

// attach session to server handle
hSession->Attach(NO);

//
// ... do something useful ...
//

// detach from session
hSession->Detach();

// perform an ISAM logout
CLISAM();

See Also

CTSession::Attach()
**CTSession::DisconnectAll**

**Syntax**

```cpp
void DisconnectAll ( )
```

**Parameters**

This method has no parameters.

**Description**

Disconnects all databases from the session.

**Return**

None.

**See also**

`Disconnect()`, `Connect()`, `CTSessionCreateDatabase()`, `Logout()`
CTSession::DropDatabase

Syntax
void DropDatabase(const CTString& Name)

Parameters
- Name [in] the database name to be dropped

Description
Drops a database from the session, but does not delete the data and index files.

Return
None.

See also
DeleteDatabase(), CreateDatabase(), AddDatabase()
**CTSession::EndImpersonation**

Disables the ability of the thread to support impersonation by another thread.

**Declaration**

```cpp
void CTSession::EndImpersonation()
```

**Description**

Finishes current impersonation (`ctImpersonateTask()`) and updates the "lock mode" on the server side with the lock mode stored before the impersonation in the `prelockmode` session property.

**Note**: This function is not supported by the c-treeACE Server by default. It requires a custom build of the server available by special request.

**Return**

```cpp
void
```

**See Also**

`CTSession::BeginImpersonation()`
CTSession::FindActive

Syntax
CTDatabase* FindActive(ULONG uid)
CTDatabase* FindActive(const CTString& Name)

Parameters
- **uid** [in] the database uid to look for in the session
- **Name** [in] the database name to look for in the session

Description
Finds an active database by uid or name.

Return
FindActive() returns a pointer to the database object or NULL if no database is found.

See also
GetFirstActive(), GetNextActive()
CTSession::FindDatabase

Syntax
CTBOOL FindDatabase(const CTString& Name, CTString& Path)
CTBOOL FindDatabase(const CTString& Name, CTString& Path, ULONG& uid)
CTBOOL FindDatabase(ULONG uid, CTString& Name, CTString& Path)

Parameters
- **Name** [in or out] the database name
- **Path** [out] the database path
- **uid** [in or out] the database uid

Description
Locates a database in a session. If the prototype `FindDatabase(CTString& Name, CTString& Path)` is used, **Name** is an input, and the method returns the database path in **Path**. If the prototype `FindDatabase(const CTString& Name, CTString& Path, ULONG& uid)` is used, **Name** is an input, and the method returns the database path, and **uid**, in **Path** and **uid**, respectively. If the prototype `FindDatabase(ULONG uid, CTString& Name, CTString& Path)` is used, **uid** is an input, and the method returns the database name, and path in **Name** and **Path** respectively.

Return
`FindDatabase()` returns YES if the database is found, NO otherwise.

See also
`FirstDatabase()`, `NextDatabase()`, `GetDatabase()`
**CTSession::FirstDatabase**

**Syntax**

```cpp
CT_BOOL FirstDatabase(CT_STRING& Name, CT_STRING& Path)
```

**Parameters**

- *Name* [out] the database name
- *Path* [out] the database path

**Description**

Gets the first database for the session.

**Return**

`FirstDatabase()` returns YES if the database is found, NO otherwise.

**See also**

`NextDatabase()`, `FindDatabase()`, `GetDatabase()`
CTSession::GetAttachMode

Declaration

CTATTACH_MODE CTSession::GetAttachMode() const;

Description

Retrieve the current session attach mode. GetAttachMode() will not execute a c-tree instance switch.

Return Values

One of the following values is returned:

<table>
<thead>
<tr>
<th>Returned CTATTACH mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTATTACH_NONE</td>
<td>Handle or Session object is not attached, or Handle not a valid session handle.</td>
</tr>
<tr>
<td>CTATTACH_SESSION</td>
<td>Handle or Session object is attached to a c-treeDB session.</td>
</tr>
<tr>
<td>CTATTACH_CTREEID</td>
<td>Handle or Session object is attached to a c-tree instance id.</td>
</tr>
<tr>
<td>CTATTACH_CURRENT</td>
<td>Handle or Session object was attached to current c-tree instance.</td>
</tr>
</tbody>
</table>

See Also

CTSession::Attach(), CTSession::Detach()
CTS::GetDatabaseCount

Syntax
NINT GetDatabaseCount();

Parameters
This method has no parameters.

Description
Retrieves the number of databases in the session dictionary. Notice that because
GetDatabaseCount() changes the "current record pointer", this method should not be used in the
middle of a search. The best approach is to use it before the initial loop to retrieve to total of
databases in the session, then use in connection with FirstDatabase() and NextDatabase() to
retrieve all database names and paths.

Return
GetDatabaseCount() returns the number of databases in the session dictionary.

See also
FirstDatabase(), NextDatabase()
CTSession::GetDatabaseUID

Syntax
void GetDatabaseUID(const CTString& Name, ULONG& uid)

Parameters
- **Name** [in] the database name
- **uid** [out] the database uid

Description
Retrieves the database UID, given the database name.

Return
None.

See also
SetDatabaseUID()
CTSession::GetFirstActive

Syntax
CTDatabase* GetFirstActive( )
CTDatabase* GetFirstActive(VRLEN& ScanRec)

Parameters
- **ScanRec** [out] the holder of the scan status

Description
Retrieves the first active database. A database is active if it is connected. The first prototype may be used when just one scan is being done. If more than one scan must be done at the same time, GetFirstActive(VRLEN& ScanRec) must be used, with different ScanRec parameters for each scan.

Return
GetFirstActive() returns the first active database in the session or 0 if there are no active databases in the session.

See also
GetNextActive(), FindActive(), CTDatabase::Connect()
CTSession::GetLibType

Retrieves the c-tree Plus operational model used when compiling the c-treeDB library.

DECLARATION
LIB_TYPE CTSession::GetLibType()

DESCRIPTION
cmdbGetLibType() provides the ability to detect c-tree Plus operational model used when compiling a c-treeDB C library.

RETURUs
ctmdbGetLibType() returns a value of type CTLIB_TYPE.

<table>
<thead>
<tr>
<th>CTLIB_TYPE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTLIB_SINGLE</td>
<td>Single user library.</td>
</tr>
<tr>
<td>CTLIB_MUSER</td>
<td>Multi-user library.</td>
</tr>
<tr>
<td>CTLIB_CLIENT</td>
<td>Client library</td>
</tr>
<tr>
<td>CTLIB_LOCLIB</td>
<td>LOCLIB model.</td>
</tr>
<tr>
<td>CTLIB_SERVER</td>
<td>Server side library.</td>
</tr>
<tr>
<td>CTLIB_THREA</td>
<td>Multi-threaded enabled.</td>
</tr>
<tr>
<td>CTLIB_TRAN</td>
<td>Transaction processing enabled.</td>
</tr>
</tbody>
</table>

The return value has the following bits set if the library supports optional features.

<table>
<thead>
<tr>
<th>Optional Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTLIB_THREA</td>
<td>Multi-threaded enabled.</td>
</tr>
<tr>
<td>CTLIB_TRAN</td>
<td>Transaction processing enabled.</td>
</tr>
</tbody>
</table>

See "c-treeDB Errors and Return Values" (page 881) for a complete listing of valid c-treeDB error codes and return values.
CTSession::GetLogonOnly

Syntax
CTBOOL GetLogonOnly( )

Parameters
This method has no parameters.

Description
Retrieves the session logon only flag. If the flag is set to YES, the user will not be able to see the session dictionary, being able to work on the tables without connection with the databases. If the flag is set to NO, the default value, the user needs a session dictionary to work with the databases or tables.

Return
GetLogonOnly() returns the session logon only flag.

See also
SetLogonOnly(), Logon()
CTSAllSession::GetNextActive

Syntax

CTDatabase* GetNextActive( )
CTDatabase* GetNextActive(VRLEN& ScanRec)

Parameters

- *ScanRec* [in and out] the holder of the scan status; it is returned by *GetFirstActive()*. 

Description

Retrieves the next active database. The first prototype may be used when just one scan is being done. If more than one scan must be done, the *GetNextActive(VRLEN& ScanRec)* must be used, with different *ScanRec* parameters for each scan. In order to use *GetNextActive()*, an active record is required. An active record is obtained using one of the retrieval methods, *GetFirstActive()* or *FindActive()*.

Return

*GetNextDatabase()* returns the next active database in the session or 0 if there are no active databases in the session.

See also

*GetFirstActive()* , *FindActive()*
CTSession::GetParam

Syntax
NINT GetParam(const CTSESSION_PARAM ParamType) const

Parameters
- **ParamType** [in] identifies the parameter type. The valid values are:
  - `CT_BUFS`
  - `CT_FILS`
  - `CT_SECT`
  - `CT_DBUFS`
  - `CT_USERPROF`

Description
Retrieves the session parameter based on the parameter type.

Return
GetParam() returns the session parameter value based on the parameter type.

See also
SetParam()
**Syntax**

```cpp
CTString GetPassword( ) const
```

**Parameters**

This method has no parameters.

**Description**

Retrieves the user password associated with the session.

**Return**

`GetPassword()` returns the user password associated with the session.

**See also**

`GetUserLogonName()`, `GetServerName()`
CTSession::GetPath

Syntax

CTString GetPath()

Parameters

This method has no parameters.

Description

Retrieves the session dictionary path (drive/directory).

Return

GetPath() returns the session dictionary path.

See also

SetPath()
CTSession::GetPathPrefix

Get the client-side path prefix.

Declaration

CTBOOL CTSession::GetPathPrefix(CTString& pathPrefix);

Description

A path prefix can be set anytime after the session handle is allocated. If a path prefix is set before a session logon, the new path prefix will affect the location of the session dictionary file. If a path prefix is set after a session logon, but before a database connect, then the path prefix affects only the database dictionary and any tables that are manipulated during that session.

If a path prefix is set, GetPathPrefix() returns YES and the path is set to pathPrefix. If no path prefix is set, GetPathPrefix() returns NO and pathPrefix is cleared.

A CException is thrown when an error occurs. Use CException::GetErrorCode() and CException::GetErrorMsg() to retrieve the error code and descriptive message.

Return

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO</td>
<td>CTSession::GetPathPrefix returns NO when no path prefix has been set.</td>
</tr>
<tr>
<td>1</td>
<td>YES</td>
<td>CTSession::GetPathPrefix returns YES when a path prefix has been returned.</td>
</tr>
</tbody>
</table>

See "c-treeDB Errors and Return Values" (page 881) for a complete listing of valid c-treeDB error codes and return values.

Example

/* get the current path prefix */
try {
    if (MySession.GetPathPrefix(MyCTStringPath))
        printf("Current path prefix is : %s
", MyCTStringPath);
    else
        printf("No current path prefix is set.\n");
}
catch(CException &E) {
    printf("ERROR: (%d) - %s\n", E.GetErrorCode(), E.GetErrorMsg());
}

See also

CTSession::SetPathPrefix()
CTSession::GetServerName

Syntax
CTString GetServerName() const

Parameters
This method has no parameters.

Description
Returns the server name associated with the session.

Return
GetServerName() returns the server name associated with the session.

See also
GetUserLogonName(), GetPassword()
CTSession::GetUserLogonName

Syntax

```cpp
CTString GetUserLogonName( ) const
```

Parameters

This method has no parameters.

Description

Returns the user name associated with the session.

Return

`GetUserLogonName()` returns the user name associated with the session.

See also

`GetServerName()`, `GetPassword()`
CTSession::IsActive

Syntax
CTBOOL IsActive( ) const

Parameters
This method has no parameters.

Description
Checks if a session is active. A session is active when the session handle has been allocated and the session is logged in to the server.

Return
IsActive() returns YES if the session is active, NO otherwise.

See also
Logon(), Logout()
CTSession::IsExclusive

Retrieves the status of the session exclusive flag.

**Declaration**

```cpp
CTBOOL CTSession::IsExclusive();
```

**Description**

`CTSession::IsExclusive()` retrieves the status of the session exclusive flag.

**Return**

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO</td>
<td>Session is not exclusive.</td>
</tr>
<tr>
<td>1</td>
<td>YES</td>
<td>Session is exclusive.</td>
</tr>
</tbody>
</table>

**Example**

```cpp
// verify an exclusive logon
CTSession hSession(CTSESSION_CTDB);

hSession.SetExclusive(YES);
hSession.Logon("FAIRCOM", "ADMIN", "ADMIN");
if (hSession.IsExclusive())
  printf("Session is exclusive\n");
else
  printf("Session is shared\n");
```

**See Also**

`CTSession::SetExclusive()`, `CTDatabase::SetExclusive()`, `CTDatabase::IsExclusive()`
**CTSession::Logon**

**Syntax**

```cpp
void Logon (const CTString& dbEngine=ctdbDEFAULT_dbengine,
            const CTString& user=ctdbDEFAULT_userid,
            const CTString& password=ctdbDEFAULT_userword)
```

**Parameters**

- **`dbEngine`**: The server or c-tree Plus instance name (default is "FAIRCOMS")
- **`user`**: The user name (default is " ")
- **`password`**: The user password (default is " ")

**Description**

Logs on to a c-tree Server, or a c-tree instance, session. A logon is necessary to work with the databases. After a session is logged on, its status is active.

**Return**

None.

**Example**

```cpp
pSession->Logon("FAIRCOMS", "ADMIN", "ADMIN");
```

**See also**

`Logout()`, `IsActive()`


## `CTSession::Logout`

### Syntax

```c
void Logout();
```

### Parameters

This method has no parameters.

### Description

Logs out from a `c-tree` Server session or from a `c-tree` instance session and release all resources. After a call to `Logout`, the handle cannot be used.

### Return

None.

### Example

```c
pSession->Logout;
```

### See also

- `Logon()`, `IsActive()`
**Syntax**

```c
CTBOOL NextDatabase(CTString& Name, CTString& Path)
```

**Parameters**

- **Name** [out] the database name
- **Path** [out] the database path

**Description**

Gets the next database for the session. A database must have already been retrieved using `FirstDatabase()`, `FindDatabase()`, or `GetDatabase()`.

**Return**

`NextDatabase()` returns YES if the database is found, NO otherwise.

**See also**

`FirstDatabase()`, `FindDatabase()`, `GetDatabase()`
CTSession::SetConfigurationFile

Stops the c-treeACE database engine .dll model.

Declaration
CTDBRET CTSession::SetConfigurationFile(void)

Description
SetConfigurationFile() sets the server .dll configuration file name and path.

Returns
Returns CTDBRET_OK on success, or throws a c-treeDB exception on failure.

Example
try {
   CTSession::SetConfigurationFile("ctsrvr.cfg");
   CTSession::StartDatabaseEngine();
} catch (CTException ex) {
   HandleException();
}...
try {
   CTSession::StopDatabaseEngine();
} catch (CTException ex) {
   HandleException();
}

See also
CTSession::StartDatabaseEngine(), CTSession::StopDatabaseEngine()
**CTSession::SetCurrentNodeName**

Sets the client node name.

**Declaration**

```cpp
void CTSession::SetCurrentNodeName(String* NodeName)
```

**Description**

When monitoring c-tree Server attached users, well written c-treeDB client applications should "register" their workstation (node) name using the standard c-tree SETNODE() (SetNodeName()) function call.

`CTSession::SetCurrentNodeName()` sets a client-side node name. `NodeName` is a string specifying the node name. The specified node name appears in the ctadmn utility under the option for "list clients". `CTSession::SetCurrentNodeName()` can only be called after a successful logon.

This functionality is available only with the c-tree Server. A call to `CTSession::SetCurrentNodeName()` on non-server environment will always return CTDBRET_OK and the node name is ignored.

**Return**

void

**Example**

```cpp
CTSession->Logon("FAIRCOMS", "ADMIN", "ADMIN");
CTSession->SetCurrentNodeName("MyNode1");
```

**See Also**

`CTIndex::SetDefault()`, `CTRecord::BuildTargetKey()`
**CTSession::SetLogonOnly**

**Syntax**

```cpp
void SetLogonOnly(CTBOOL flag = YES)
```

**Parameters**

- `flag` [in] the new session logon flag.

**Description**

Sets the session logon only flag. This flag, when set to YES before the session Logon, will prevent the session from using the session dictionary.

**Return**

None.

**See also**

GetLogonOnly()
**CTSession::SetExclusive**

Sets or clears the session exclusive flag.

**Declaration**

```cpp
void CTSession::SetExclusive(CTBOOL flag);
```

**Description**

- If `flag` is set to YES it will set the exclusive flag, while NO will clear the exclusive flag.

The `CTSession::SetExclusive()` sets or clears the session exclusive flag. If a session exclusive flag is set, only one `CTSESSION_CTDB` or `CTSESSION_SQL` session will be allowed. Set the session exclusive flag after instantiating a `CTSession` object, but before performing a logon. Setting the session exclusive flag after a session logon is performed will not have any effect during the current session.

**Return**

`void`

**Example**

```cpp
// perform an exclusive logon
CTSession hSession(CTSESSION_CTDB);

hSession.SetExclusive(YES);
hSession.Logon("FAIRCOM", "ADMIN", "ADMIN");
```

**See Also**

`CTSession::IsExclusive()`, `CTDatabase::SetExclusive()`, `CTDatabase::IsExclusive()`
CTSession::SetParam

Syntax
void SetParam(const CTSESSION_PARAM ParamType, const NINT value)

Parameters
- **ParamType** [in] identifies the parameter type. The valid values are:
  - CT_BUFS
  - CT_FILS
  - CT_SECT
  - CT_DBUFS
  - CT_USERPROF
- **value** [in] parameter value

Description
Sets the session parameter values.

Return
None.

See also
GetParam()
CTSession::SetPath

Syntax
void SetPath(const CTString Path)

Parameters
- Path [in] the session dictionary path.

Description
Sets the session dictionary path.

Return
None.

See also
getPath(), Create()
CTSession::SetPathPrefix

Set the client-side path prefix.

Declaration

void CTSession::SetPathPrefix(const CTString& pathPrefix);
void CTSession::SetPathPrefix();

Description

A path prefix can be set anytime after the session handle is allocated. If a path prefix is set before a session logon, the new path prefix will affect the location of the session dictionary file. If a path prefix is set after a session logon, but before a database connect, then the path prefix affects only the database dictionary and any tables that are manipulated during that session.

A CException is thrown when an error occurs. Use CException::GetErrorCode() and CException::GetErrorMsg() to retrieve the error code and descriptive message.

Return

void

Example

/* set a path prefix */
try {
    MySession.SetPathPrefix("C:\FairCom\V8.14\ctreeSDK\ctreeAPI");
} catch(CException &E) {
    printf("SetPathPrefix error: (%d) - %s\n", E.GetErrorCode(),
            E.GetErrorMsg());
}

/* clear a path prefix */
try {
    MySession.SetPathPrefix("");
} catch(CException &E) {
    printf("SetPathPrefix error: (%d) - %s\n", E.GetErrorCode(),
            E.GetErrorMsg());
}

See Also

CTSession::GetPathPrefix()
**CTSession::StartDatabaseEngine**

Starts the c-treeACE database engine .dll model.

**Declaration**

```cpp
CTDBRET CTSession::StartDatabaseEngine(void)
```

**Description**

A developer can call `CTSession::StartDatabaseEngine()` once before calling any c-treeDB methods, and then call function `CTSession::StopDatabaseEngine()` when finished with database activities to shutdown the database engine. `CTSession::StartDatabaseEngine()` must be called before any other c-treeDB methods are performed. `CTSession::StartDatabaseEngine()` loads the `ctsrvr.cfg` configuration file, creates the data and index caches, and initializes the multithreading subsystem.

Note that automatic recovery takes place on startup which can result in short delays after making this call, especially if the process was interrupted previously and an application shutdown without calling `CTSession::StopDatabaseEngine()`.

This is a static method and it can be invoked without the presence of a `CTSession` object.

**Returns**

`StartDatabaseEngine()` returns CTDBRET_OK on success, or throws a c-treeDB exception on failure.

**See also**

`CTSession::StopDatabaseEngine()`, `CTSession::SetConfigurationFile()`
`CTSession::StopDatabaseEngine` stops the c-treeACE database engine .dll model.

**Declaration**

```cpp
CTDBRET CTSession::StopDatabaseEngine(void)
```

**Description**

`CTSession::StopDatabaseEngine()` shuts down the c-treeACE database engine .dll model. This ensures a clean shutdown, resulting in quicker startups at the next application instance. It also ensures all files are properly closed and flushed to disk.

This is a static method and it can be invoked without the presence of a `CTSession` object.

**Returns**

Returns CTDBRET_OK on success, or throws a c-treeDB exception on failure.

**See also**

`CTSession::StartDatabaseEngine()`, `CTSession::SetConfigurationFile()`
4.18 CTString Class

class CTString

Description
The CTString class represents String objects.

See also
CTChar

Preconditions
This is one of the basic objects from the c-tree database layer.

CTString Operators
- =, +, += : Assignment and concatenation operators
- [] : Reference to a character operator
- <, <=, >, >=, ==, != : Comparison operators
CTString::operator =, +, +=

Syntax

CTString& operator=(const CTString& str)
CTString& operator=(const pTEXT str)
CTString& operator+=(const CTString& str)
CTString operator+(const CTString& lhs, const CTString& rhs)
CTString operator+(const pTEXT lhs, const CTString& rhs)

Parameters

- **str** [in] The *CTString* object or text to be assigned or concatenated to form the new *CTString* object
- **lhs** [in] The left hand side string or text to be concatenated
- **rhs** [in] The right hand side string to be concatenated

Description

Assigns or concatenates a string or text to form a *CTString* object

Return

The new *CTString* object
**CTString::operator[]**

**Syntax**

```cpp
char& operator[](const NINT index)
```

**Parameters**

- `index [in]` The index to search

**Description**

Returns a reference to the character at the index position.

**Return**

Returns a reference to the character at the index position, or throws an exception if index<0 or index>length of string.
CTString::operator<, <=, >, >=, ==, !=

Syntax
CTBOOL operator<(const CTString& str)
CTBOOL operator<=(const CTString& str)
CTBOOL operator>(const CTString& str)
CTBOOL operator>=(const CTString& str)
CTBOOL operator==(const CTString& str)
CTBOOL operator!=(const CTString& str)

Parameters
- str [in] The string to be compared with the CTString object

Description
Compares the CTString object with str.

Return
Returns YES or NO depending on the result of the comparison.
CTString Methods

Constructor / Destructor
- **CTString()**: Creates a `CTString` object
- **~CTString()**: Destroys a `CTString` Object and resets all the dependent objects

String Handling
- **c_str()**: Returns a ‘C’ language format character string, with the same contents represented by the `CTString` object
- **Compare()**: Compares this `CTString` object to str, with case sensitivity
- **CompareIC()**: Compares this `CTString` object to str, with case insensitivity
- **Delete()**: Deletes characters from the string
- **Empty()**: Frees the string and sets it to NULL
- **Insert()**: Inserts `substr` into this string at position index
- **IntToHex()**: Converts a number into a string containing the number’s hexadecimal representation
- **IsDelimiter()**: Verifies if the character at byte index in the `CTString` matches any character in the delimiters the string str
- **IsEmpty()**: Verifies if this string is NULL or the length is zero.
- **IsPathDelimiter()**: Verifies if the character at byte index in the `CTString` is ‘\’ (or ‘/’ for Unix systems)
- **LastChar()**: Retrieves a pointer to the last full character in the `CTString`
- **LastDelimiter()**: Retrieves the byte index of the rightmost whole character in this `CTString`
- **Left()**: Retrieves the count left most characters in the string
- **Length()**: Retrieves the length of the string
- **LowerCase()**: Converts this `CTString` characters to lower case
- **PadLeft()**: Pads this `CTString` to the left
- **PadRight()**: Pads this `CTString` to the right
- **Pos()**: Returns the index at which a specified substring begins
- **Right()**: Retrieves the count right most characters in the string
- **SetLength()**: Resizes this `CTString`, truncating or increasing the size of the string
- **StringOfChar()**: Fills a `CTString` with the specified number of characters
- **SubString()**: Retrieves a specified substring of the `CTString`
- **ToInt()**: Converts a `CTString` to an integer
- **ToDouble()**: Converts a `CTString` to a double
- **Trim()**: Removes all spaces from the beginning and from the end of a `CTString`
- **TrimLeft()**: Removes all spaces from the beginning of a `CTString`
- **TrimRight()**: Removes all spaces from the end of a `CTString`
- **UpperCase()**: Converts this `CTString` characters to upper case
**CTString::CTString**

**Syntax**

CTString
CTString(const pTEXT ptr)
CTString(const pTEXT ptr, VRLEN size)
CTString(const CTString& str)
CTString(const NINT value)
CTString(const double value)

**Parameters**

- **ptr** [in] The pointer to a C String.
- **size** [in] The size of the string.
- **str** [in] The string to be copied in the copy constructor.
- **value** [in] A NINT value to be converted to a CTString

**Description**

This is the constructor for the CTString object. The prototype CTString(const CTString& str) is the copy constructor for the CTString class.

**See also**

~CTString()
CTString::~CTString

Syntax
~CTString

Parameters
This destructor has no parameters.

Description
This is the destructor for the CTString object.

See also
CTString()}
**Syntax**
```cpp
pTEXT c_str() const
```

**Parameters**
This method has no parameters.

**Description**
c_str returns a 'C' language format character string, with the same contents represented by the `CTString` object.

**Return**
c_str returns a pointer with a null-terminated character array containing the same characters as the `CTString` class. If the `CTString` is unassigned, c_str returns NULL.

**Example**
```cpp
CString name = "John";
printf("%s", CTString_sample.c_str());
```
CTString::Compare

Syntax
NINT Compare(const CTString& str) const

Parameters

- `str [in]` The string to be compared.

Description

Compares this `CTString` object to `str`, with case sensitivity. This method is NULL safe.

Return

Compare returns:

- A value greater than zero if the `CTString` object is greater than `str`
- A value smaller than zero if the `CTString` object is smaller than `str`
- Zero if they are the same

See also

`CompareIC()`
CTString::CompareIC

Syntax
NINT CompareIC(const CTString& str) const

Parameters
- str [in] The string to be compared.

Description
Compares this CTString object to str, with case insensitivity. This method is NULL safe.

Return
CompareIC() returns:
- A value greater than zero if the CTString object is greater than str
- A value smaller than zero if the CTString object is smaller than str
- Zero if they are the same

See also
Compare()
CTString::Delete

Syntax
void Delete(const NINT index, const NINT count)

Parameters
- **index** [in] The position in the string where characters will be deleted
- **count** [in] The number of characters to be deleted

Description
Deletes count characters from the string, starting at character position index.

Return
None.

See also
Empty(), Insert()
CTString::Empty

Syntax
void Empty( )

Parameters
This method has no parameters.

Description
Frees the string and sets it to NULL.

Return
None.

Example
CTString database_name;
database_name.Empty();
memset(buffer, ' ', sizeof(buffer));
printf("\n\nEnter Database name: ");
gets(buffer);
if (strlen(buffer) > 0)
{
    database_name = buffer;
}

See also
Delete(), IsEmpty(), SetLength()
CTString::Insert

Syntax
void Insert(const CTString& substr, const NINT index)

Parameters
- substr [in] The substring object to be inserted in the CTString object
- index [in] The position in this string where substr will be inserted.

Description
Inserts substr into this string at position index. The first character position is 0. If index is zero
Insert is similar to *this = substr + *this. If index is equal to Length then Insert is similar to *this =
*this + substr. If index is less than zero or greater than Length, a CException is thrown with error
code set to CTDBRET_INDEXRANGE (4005). If substr is empty, no operation takes place.

Return
None.

See also
Delete()
CTString::IntToHex

Syntax
CTString IntToHex(const NINT value, const NINT digits)

Parameters
- value [in] The integer value to be converted to hexadecimal.
- digits [in] The minimum number of hexadecimal digits. The maximum acceptable value of
digits is 256.

Description
Converts a number into a string containing the number’s hexadecimal (base 16) representation.

Return
IntToHex() returns a CTString object with the hexadecimal representation of value.

See also
ToInt()
CTString::IsDelimiter

Syntax
CTBOOL IsDelimiter(const CTString& str, const NINT index) const

Parameters
- str [in] The string with the delimiter characters.
- index [in] The position in string to check for delimiters. The first position is zero. If index is less than zero or if index is greater or equal to the length of the string, a CTException is thrown with error code CTDBRET_INDEXRANGE (4005).

Description
Verifies if the character at byte index in the CTString matches any character in the delimiters the string str.

Return
IsDelimiter() returns YES if the character at byte index in the CTString matches any character in the delimiters string str, NO otherwise.

See also
IsPathDelimiter()
CTString::IsEmpty

Syntax

CTBOOL IsEmpty( ) const

Parameters

This method has no parameters.

Description

Verifies if this string is NULL or the length is zero.

Return

IsEmpty() returns YES if the string is NULL or length is zero, NO otherwise.

See also

Empty(). Length()
CTString::IsPathDelimiter

Syntax
CTBOOL IsPathDelimiter(const NINT index)

Parameters
- index [in] The position in string to check for path delimiter. The first position is zero. If index is less than zero or if index is greater or equal to the length of the string, a CTException is thrown with error code CTDBRET_INDEXRANGE (4005).

Description
Verifies if the character at byte index in the CTString is `\` (or `/` for Unix systems).

Return
IsPathDelimiter() returns YES if the character at byte index in the CTString is a path delimiter, NO otherwise.
CTString::LastChar

Syntax
pTEXT LastChar( ) const

Parameters
This method has no parameters.

Description
Retrieves a pointer to the last full character in the CTString.

Return
LastChar() returns a pointer to the last full character in the CTString

See also
LastDelimiter()
CTString::LastDelimiter

Syntax
NINT LastDelimiter(const CTString& delimiters) const

Parameters
- `delimiters` [in] string with the delimiters.

Description
Retrieves the byte index in this `CTString` of the rightmost whole character that matches any character in delimiters, except null (\0).

Return
`LastDelimiter()` returns the byte index of the rightmost whole character in this `CTString`.

See also
`LastChar()`, `IsDelimiter()`
**CTStringRef::Left**

**Syntax**

CTStringRef Left(const NINT count)

**Parameters**

- **count [in]** The number of characters to be retrieved.

**Description**

Retrieves the count left most characters in the string. If count is greater than the length of the string, the whole string is returned.

**Return**

Left returns a *CTStringRef* object with the count left most characters.

**See also**

Right()
CTString::Length

Syntax
NINT Length( ) const

Parameters
This method has no parameters.

Description
Retrieves the length of the string.

Return
Length returns the length of the string. Returns zero if the string is NULL or empty.

See also
IsEmpty(), SetLength()
CTString::LowerCase

Syntax
void LowerCase( )

Parameters
This method has no parameters.

Description
Converts this CTString characters to lower case.

Return
None.

See also
UpperCase()
**CTString::PadLeft**

**Syntax**
void PadLeft(const NINT newlength, const char padChar= ' ')

**Parameters**
- `newlength` [in] The length of the padded string. If `newlength` is less than this `CTString`, no padding is performed.
- `padChar` [in] The character used to pad the string. By default, ' ' is used.

**Description**
Pads this `CTString` to the left with `padChar` characters until the length of this `CTString` is equal to `newlength`.

**Return**
None.

**See also**
- `PadRight()`
**CTString::PadRight**

Syntax

```cpp
void PadRight(const NINT newlength, const char padChar = ' ')
```

Parameters

- `newlength` [in] The length of the padded string. If `newlength` is less than this `CTString`, no padding is performed.
- `padChar` [in] The character used to pad the string. By default, `' '` is used.

Description

Pads this `CTString` to the right with `padChar` characters until the length of this `CTString` is equal to `newlength`.

Return

None.
**CTString::Pos**

**Syntax**

```cpp
NINT Pos(const CTString& substr) const
```

**Parameters**

- `substr [in]` The string to look for.

**Description**

Retrieves the index at which a specified substring begins.

**Return**

- `Pos()` returns the index number at which a specified substring begins or -1 if `substr` is not found.

**See also**

- `SubString()`
CTString::Right

Syntax
CTString Right(const NINT count)

Parameters
- count [in] The number of characters to be retrieved.

Description
Retrieves the count right most characters in the string. If count is greater than the length of the string, the whole string is returned.

Return
Right() returns a CTString object with the count right most characters.

See also
Left()
CTString::SetLength

Syntax
void SetLength(const NINT newLength)

Parameters
- `newLength` [in] The new length of the string. If `newLength` is zero, the string is cleared (set to NULL).

Description
Resizes this CTString. SetLength() truncates or increases the size of the string.

Return
None.

See also
Length(), Empty()
CTString::StringOfChar

Syntax
CTString StringOfChar(const char ch, const NINT count)

Parameters
- \textit{ch [in]} Character used to fill the string.
- \textit{count [in]} The number of times to repeat the character.

Description
Fills a \textit{CTString} with the specified number of characters.

Return
\texttt{StringOfChar()} returns the \textit{CTString} object filled with the specified character.
CTString::SubString

Syntax
CTString SubString(const NINT index, const NINT count) const

Parameters
- **index** [in] The start position of the substring. An empty string will be returned if index is greater than the length of the CTString. If index is negative, a CTE exception is thrown with the error code set to CTDBRET_INDEXRANGE (4005).
- **count** [in] The maximum number of characters in the substring.

Description
Retrieves a specified substring of the CTString.

Return
SubString returns the CTString object with the substring.

See also
Pos()
CTString::ToInt

Syntax
NINT ToInt( ) const

Parameters
This method has no parameters.

Description
Converts a CTString to an integer.

Return
ToInt() returns an integer value converted from CTString. ToInt() returns zero if the CTString does not contain a valid integer.

See also
IntToHex(), ToDouble()
**CTString::ToDouble**

**Syntax**

double ToDouble( ) const

**Parameters**

This method has no parameters.

**Description**

Converts a *CTString* to a double.

**Return**

`ToDouble()` returns a double value converted from *CTString*. `ToDouble()` returns 0.0 if the *CTString* does not contain a valid floating point number.

**See also**

`ToInt()`
CTString::Trim

Syntax
void Trim();

Parameters
This method has no parameters.

Description
Removes all spaces from the beginning and from the end of a CTString.

Return
None.

See also
TrimLeft(). TrimRight()
CTString::TrimLeft

Syntax
void TrimLeft();

Parameters
This method has no parameters.

Description
Removes all spaces from the beginning of a CTString.

Return
None.

See also
Trim(), TrimRight()
CTString::TrimRight

Syntax
void TrimRight();

Parameters
This method has no parameters.

Description
Removes all spaces from the end of a CTString.

Return
None.

See also
Trim(), TrimLeft()
CTString::UpperCase

Syntax
void UpperCase( )

Parameters
This method has no parameters.

Description
Converts this CTString characters to upper case.

Return
None.

See also
LowerCase()
4.19 CTTable Class

class CTTable

Description
Objects of the CTTable class are used to manage tables. One table may belong to several databases, and a database may have multiple tables. The table may have any number of fields. It uses CTBase as the base class, and implements the constructor/destructor allocating/freeing memory to the table operations.

See also
CTBase, CTDatabase, CTRecord

Preconditions
Before performing operations with a CTTable object, it is necessary to instantiate session and database objects.
CTTable Methods

Constructor / Destructor

- **CTTable()**: Creates a **CTTable** object
- ~CTTable(): Destroys a **CTTable** Object and resets all the dependent objects

Table Handling

- **Alter()**: Performs alter table functions
- **Close()**: Closes a table
- **Create()**: Creates a new table
- **GetCreateMode()**: Retrieves the table create mode.
- **GetDataDefaultExtentSize()**: Retrieves the c-tree data file default extent size
- **GetDataExtension()**: Retrieves the data file name extension.
- **GetDatno()**: Retrieve the table data file number.
- **GetGroupid()**: Retrieves the table group id
- **GetIndexDefaultExtentSize()**: Retrieves the c-tree index file default extent size
- **GetIndexExtension()**: Retrieves the index file name extension.
- **GetIdxno()**: Retrieve a table index file number.
- **GetName()**: Retrieves the table name, without the name extension.
- **GetOpenMode()**: Retrieves the table open mode.
- **GetPassword()**: Retrieves the table password
- **GetPath()**: Retrieves the table path.
- **GetPermission()**: Retrieves the table permission mask
- **HasNullFieldSupport()**: Indicates if a table has null field support
- **HasDelField()**: Indicates if a table has delete field support.
- **IsActive()**: Indicates if a table is open or closed.
- **Open()**: Opens an existing table
- **ResetAll()**: Resets all active record buffers associated with the table
- **SetDataDefaultExtentSize()**: Sets the c-tree data file default extent size
- **SetDataExtension()**: Sets the table data file extension
- **SetGroupid()**: Sets the table groupid
- **SetIndexDefaultExtentSize()**: Sets the c-tree index file default extent size
- **SetIndexExtension()**: Sets the table index file extension.
- **SetPassword()**: Sets the table password
- **SetPath()**: Sets the table path
- **SetPermission()**: Sets the table permission mask
- **UnlockTable()**: Unlocks all record locks from the table.
- **UpdateCreateMode()**: updates the table create mode
Field Handling
- **AddField()**: Adds a new field to the table.
- **DelField()**: Deletes one field from the table.
- **GetField()**: Retrieves a field by its number or name.
- **GetFieldCount()**: Retrieves the number of fields in the table.
- **GetFieldName()**: Retrieves the field number from a field name.
- **GetPadChar()**: Retrieves the table pad and field delimiter characters.
- **InsertField()**: Inserts a new field to the table, in the specified position.
- **MoveField()**: Moves the field from one position to a new position in the table.
- **SetPadChar()**: Sets the table pad and field delimiter characters.
- **UpdatePadChar()**: Updates the table pad and delimiter character resource.

Filter Handling
- **FilterRecord()**: Sets the filtering for a table.
- **GetFilter()**: Retrieves the filter that is filtering the table.
- **IsFilteredRecord()**: Indicates if the table is being filtered or not.

Index Handling
- **AddIndex()**: Adds a new index to the table
- **DelIndex()**: Deletes an index from the table
- **GetCndxIndex()**: Retrieves the conditional index expression string, given the index number.
- **GetCndxIndexByName()**: Retrieves the conditional index expression string, given the index name.
- **GetCndxIndexLength()**: Retrieves the length in bytes of the conditional expression string, given the index number.
- **GetCndxIndexLengthByName()**: Retrieves the length in bytes of the conditional expression string, given the index name.
- **GetIndex()**: Retrieves an index object
- **GetIndexByUID()**: Retrieves the index given its UID
- **GetIndexCount()**: Retrieves the number of indices of the table
- **HasRecbyt()**: Indicates if a table was created with support to a recbyt index
- **HasRowid()**: Indicates if a table was created with support to a rowid index

Segment Handling
- **AddSegment()**: Adds a new segment to an index
- **DelSegment()**: Deletes an existing segment
- **GetSegment()**: Retrieves an index segment object
- **InsertSegment()**: Inserts a new segment
CTTable::CTTable

Syntax
CTTable(const CTDatabase& Handle)
CTTable(const CTDatabase* Handle)
CTTable(const CTSession& Handle)
CTTable(const CTSession* Handle)

Parameters
- **Handle** [In] Is the database or session object. When the `CTTable(const CTDatabase& Handle)` or the `CTTable(const CTDatabase* Handle)` prototype is used, **Handle** is a database object or pointer to a database object. When the `CTTable(const CTSession& Handle)` or the `CTTable(const CTSession* Handle)` prototype is used, **Handle** is a session object or pointer to a session object. The **CTTable** object being created is a child of the **CTDatabase** or **CTSession** object.

Description
This is the constructor for the **CTTable** class.

See also
~CTTable()
**CTTable::~CTTable**

**Syntax**

~CTTable

**Parameters**

This destructor has no parameters.

**Description**

This is the destructor for the `CTTable` class. It frees the allocated memory to the Table. No table operations are allowed after the `CTTable` destructor is called.

**See also**

`CTTable()`
**CTTable::AddField**

**Syntax**

```cpp
CTField AddField(const CTString& Name, CTDBTYPE Type, VRLEN Length)
```

**Parameters**

- **Name** [in] - The field name to add to the table
- **Type** [in] - The field type. Valid field type values are shown in the "Field Types" (page 227).
- **Length** [in] - The field length, the maximum length of [VAR]CHAR or [VAR]BINARY field in bytes (64K max). Set to 0 for "unlimited" (2GB) LVARCHAR.

**Description**

Adds a new field to table.

**Return**

`AddField()` returns a field object on success or NULL on failure.

**See also**

`InsertField()`, `DelField()`, `GetField()`, `AddIndex()`
**CTTable::AddIndex**

**Syntax**

```cpp
CTIndex AddIndex(const CTString& name, CTDBKEY KeyType,
                  CTBOOL AllowDuplicates, CTBOOL NullFlag)
```

**Parameters**

- **name** [in] The index name to add to the table
- **KeyType** [in] The index type. Allowed types are:
  - CTINDEX_FIXED
  - CTINDEX_LEADING
  - CTINDEX_PADDING
  - CTINDEX_LEADPAD
  - CTINDEX_ERROR
- **AllowDuplicates** [in] The flag indicating if the index accepts duplicates or not.
- **NullFlag** [in] The flag indicating if Null fields are allowed or not.

**Description**

Adds a new index to the table

**Return**

None.

**See also**

`DelIndex()`, `AddField()`, `HasNullFieldSupport()`
### CTTable::AddSegment

**Syntax**

```cpp
CTSegment AddSegment(const CTIndex& pIndex, const CTField& pField, CTSEG_MODE SegMode)
CTSegment AddSegment(const CTIndex& pIndex, const CTString& FieldName, CTSEG_MODE SegMode)
CTSegment AddSegment(const CTIndex& pIndex, NINT FieldNumber, CTSEG_MODE SegMode)
CTSegment AddSegment(const CTIndex& pIndex, NINT offset, NINT Length, CTSEG_MODE SegMode)
CTSegment AddSegment(NINT IndexNbr, NINT FieldNbr, CTSEG_MODE SegMode)
CTSegment AddSegment(NINT IndexNbr, const CTString& FieldName, CTSEG_MODE SegMode)
CTSegment AddSegment(NINT IndexNbr, const CTField& pField, CTSEG_MODE SegMode)
CTSegment AddSegment(const CTString& IndexName, const CTString& FieldName, CTSEG_MODE SegMode)
CTSegment AddSegment(const CTString& IndexName, NINT FieldNumber, CTSEG_MODE SegMode)
CTSegment AddSegment(const CTString& IndexName, const CTField& pField, CTSEG_MODE SegMode)
CTSegment AddSegment(const CTString& IndexName, const CTString& FieldName, const CTField& pField, CTSEG_MODE SegMode)
```

**Parameters**

- `pIndex` [in] A pointer to the index object.
- `pField` [in] A pointer to the field object that composes part of the index segment
- `SegMode` [in] The segment mode. The valid values for the segment mode are shown in "Segment Modes" (page 92). Notice that, if this method is used in the implementation that requires the offset, the segment is called extended, and in this specific case, the segment modes should be those that depend on the offset. If the implementation used is any other, the segment mode should be one of the so called schema segments (CTSEG_SCHSEG, CTSEG_USCHSEG, CTSEG_VSCHSEG, CTSEG_UVSCHSEG, CTSEG_SCHSRL).
- `FieldName` [in] Field name to compose the index segment
- `FieldNumber` [in] Field number to compose the index segment
- `offset` [in] Absolute byte offset of the segment. If used, this offset should be calculated with care, since the hidden fields included by c-treeDB should be taken into account: $DELFLD$ (always included, to account for the deleted record) - 4 bytes; $NULFLD$ (always present - 1 bit for every user defined field, rounded for the next byte); $ROWID$ (8 bytes to account for the automatic auto-increment record - see the discussion on ROWID in "Hidden fields" (page 89)). This variation makes it difficult to predict the correct offset of each user defined offset.
- `Length` [in] Segment length in bytes
- `IndexNbr` [in] Index number
- `FieldNbr` [in] Field number to compose the index segment
- `IndexName` [in] Index name to be updated.

**Description**

Adds a new segment to an index.

It is strongly recommended that the `CTSegment::AddSegment(const CTIndex& pIndex, NINT offset, NINT Length, CTSEG_MODE SegMode)` implementation is not used, with considerable advantages for the use of the other implementations. The use of the extended segments (based
on the segment offset), as needed in this implementation, may prevent the use of advanced
ctreeDB functions like \texttt{CTTable::Alter()}. If, for any reason, it is mandatory to use this
implementation, try to adapt the example presented in the description of the
\texttt{ctdbAddSegmentEx()} function.

\textbf{Return}

\texttt{AddSegment()} returns a segment object.

\textbf{See also}

\texttt{InsertSegment()}, \texttt{DelSegment()}
CTTable::Alter

Syntax
void Alter(NINT Action)

Parameters
- **Action** [in] The alter table action. One or more of the following actions can be selected:
  - `CTDB_ALTER_NORMAL`: Check for changes before altering
  - `CTDB_ALTER_INDEX`: Force rebuild of all indices
  - `CTDB_ALTER_FULL`: Force full table rebuild.
  - `CTDB_ALTER_TRUNCATE`: Quickly remove all records.
  - `CTDB_ALTER_PURGEDUP`: OR in with any of the other modes to deal with cases where an alter table is performed on data files with duplicated index information.
  - `CTDB_ALTER_KEEPTRAN`: Allow tables with different TRANDEP modes (e.g. transaction capable, however, transaction control disabled for performance) to undergo ALTER TABLE. (This is generally an internal FairCom operation for certain table types.)

Description
Performs alter table functions. This method may be used to update a table after its creation. The table must be open in `CTOPEN_EXCLUSIVE` mode before it is updated.

Return
None.

See also
Open()
CTTable::Attach

Attaches a c-tree Plus ISAM datno object to a c-treeDB table object.

Declaration
void CTTable::Attach(NINT datno);

Description
- datno is data file number opened with one of the c-tree ISAM open functions.

CTTable::Attach() attaches a c-tree Plus ISAM datno object to a c-treeDB table object. This function is useful if you have opened a data and index file using one of c-tree's ISAM open functions and need to attach it to a table handle to use some of the advanced c-treeDB features such as alter table or the record handler.

Return
void

Example

CTSession hSession(CTSESSION_CTREE);
CTTable hTable(hSession);
CTRecord hRecord(hTable);
NINT datno, count = 0;

// logon to c-tree
hSession.Logon(SERVER, USER, PASSWD);

// open the file using c-tree ISAM
datno = (NINT)OPNRFILX((COUNT) -1, "test309.dat", (COUNT)0, NULL);

// attach to table
hTable.Attach(datno);

// read the records
if (hRecord.First())
do {
    count++;
} while (hRecord.Next());

// cleanup
hTable.Detach();
hSession.Logout();

See Also
CTTable::AttachXtd(), CTTable::Detach()
CTTable::AttachXtd

Attaches a c-tree Plus ISAM datno object to a c-treeDB table handle allowing a DODA and IFIL to be specified.

Declaration

```cpp
void CTTable::AttachXtd(NINT datno, NINT nbrfields, pDATOBJ dodaptr,
                      pIFIL ifilptr);
```

Description

- `datno` is the data file number.
- `nbrfields` is the number of fields described by `dodaptr`. If `nbrfields` is zero, it is assumed that the DODA is not supplied and a DODA resource will be read from the table itself.
- `dodaptr` pointer to DODA entries for each field in table. If `dodaptr` is NULL, it is assumed that the DODA is not supplied and a DODA resource will be read from the table itself.
- `ifilptr` is a pointer to IFIL structure describing indices of table. If `ifilptr` is NULL it is assumed that an IFIL is not supplied and an IFIL resource will be read from the table itself.

CTTable::AttachXtd() attaches a c-tree Plus ISAM datno object to a c-treeDB table object. This function is useful if you have opened a data and index file using one of c-tree’s ISAM open functions and need to attach it to a table handle to use some of the advanced c-treeDB features such as alter table or the record handler. This extended version allows the user to specify the DODA and IFIL for the table, enabling tables without DODA and/or IFIL to be attached to c-treeDB.

Return

void

Example

```cpp
/* DODA */
static DATOBJ doda[] =
{   
  {"f1", (pTEXT)0, CT_INT4, 4},
  {"f2", (pTEXT)4, CT_FSTRING, 10}
};
/* IFIL */
static ISEG iseg = {0, 4, 12};
static IIDX iidx = {4, 0, 0, 0, 0, 1, &iseg, "i311x1", NULL, NULL, NULL};
static IFIL ifil = {"test310", -1, 14, 0, 0, 1, 0, &iidx, "f1", "f2", 0};

CTSession hSession(CTSESSION_CTREE);
CTTable hTable(hSession);
CTRecord hRecord(hTable);
NINT datno, count = 0;

// logon to c-tree
hSession.Logon(SERVER, USER, PASSWD);
// open the file using c-tree ISAM
datno = (NINT)OPNRFILX((COUNT) -1, "test309.dat", (COUNT)0, NULL);
// attach to table
hTable.AttachXtd(datno, 2, doda, &ifil);
// read the records
```
if (hRecord.First())
do{
    count++;
}while (hRecord.Next());
// cleanup
hTable.Detach();
hSession.Logout();
CTTable::ClearAllFieldDefaultValue

Clears the default value associated with all fields in a table.

Declaration

void CTTable::ClearAllFieldDefaultValue();

Description

Clears the default value associated with all fields in a table. The default date and time types for each field are also reset to the default values of `CTDATE_MDCY` and `CTIME_HMS` respectively.

Return

void

Example

// clear all field default value
try
{
  hTable.ClearAllFieldDefaultValue();
} catch (CTException &err)
{
  printf("ClearAllFieldDefaultValue failed\n");
}

See Also

`SetFieldDefaultValue()`, `GetFieldDefaultValue()`, `IsFieldDefaultValueSet()`, `ClearFieldDefaultValue()`, `SetFieldDefaultDateTimeType()`, `GetFieldDefaultDateType()`, `GetFieldDefaultTimeType()`
CTTable::Clone

Syntax
void Clone(CTTable &target, const CTString& Name)

Parameters
- target [in] - The new table handle
- Name [in] - The new table name (without file extension)

Description
Creates a new table with the attributes from an existing source table. **This does NOT create a duplicate table -- records are not copied.**

**Note:** This function is only for a `CTSESSION_CTREE` session or for cloning tables from/into different databases.

Return
None. Throws a `CTException` on error. Typical `CreateTable()` exceptions are possible.

See also
CTTable::CreateTable()
CTTable::Close

Syntax
void Close( )

Parameters
This method has no parameters.

Description
Closes a table.

Return
None.

See also
Open()
CTTable::Create

Syntax
void Create(const CTString& Name, const CTCREATE_MODE CreateMode)

Parameters
- **Name** [in] The table name
- **CreateMode** [in] The create mode for the table. The valid values for the table create mode are shown in "Table Create Modes" (page 234).

Description
Creates a new table. Before the table creation, it must be defined with AddField(), and optionally with AddIndex() and AddSegment().

Return
None.

See also
AddField(), GetCreateMode(), Open()
CTTable::DelField

Syntax
void DelField(const NINT FieldNumber)
void DelField(const CTString&FieldName)

Parameters

- **FieldNumber** [in] Delete this field number, if the prototype `DelField(const NINT FieldNumber)` is used.
- **FieldName** [in] Delete this field name, if the prototype `DelField(const CTString& FieldName)` is used.

Description

Deletes a field from the table.

Return

None.

See also

InsertField(), AddField(), CTDatabase::DeleteTable()
CTTable::DelIndex

Syntax
void DelIndex(NINT IndexNumber)
void DelIndex(const CTString& IndexName)

Parameters
- **IndexNumber** [in] Index number to be deleted, if the prototype `DelIndex(NINT IndexNumber)` is used.
- **IndexName** [in] Index name to be deleted, if the prototype `DelIndex(const CTString& IndexName)` is used.

Description
Deletes an index from a table, either by name or number.

Return
None.

See also
`AddIndex()`, `DelField()`
CTTable::DelSegment

Syntax
void DelSegment(NINT IndexNumber, NINT SegNumber)
void DelSegment(const CTString& IndexName, NINT SegNumber)

Parameters
- \textit{IndexNumber} [in] The index number whose segment is to be deleted.
- \textit{SegNumber} [in] The segment number to be deleted.
- \textit{IndexName} [in] The index name whose segment is to be deleted.

Description
Deletes an existing segment.

Return
None.

See also
AddSegment(), InsertSegment()
CTTable::Detach

Detaches a c-treeDB table object from a c-tree data and index files.

Declaration

void CTTable::Detach();

Description

CTTable::Detach() detaches a c-treeDB table object from a c-tree data and index files. The table is not closed but the c-treeDB table handle resources are released and the handle re-initialized.

Return

void

Example

CTSession hSession(CTSESSION_CTREE);
CTTable hTable(hSession);
CTRecord hRecord(hTable);
NINT datno, count = 0;

// logon to c-tree
hSession.Logon(SERVER, USER, PASSWD);

// open the file using c-tree ISAM
datno = (NINT)OPNRFILX((COUNT)-1, "test309.dat", (COUNT)0, NULL);

// attach to table
hTable.Attach(datno);

// read the records
if (hRecord.First())
    do
        count++;
    while (hRecord.Next());

// cleanup
hTable.Detach();
hSession.Logout();

See Also

CTTable::Attach(), CTTable::AttachXtd()
CTTable::FilterRecord

Syntax
void FilterRecord(const CTString& cndexpr)

Deprecated
Returns CTDBRET_NOTSUPPORTED. c-treeDB Filters are record based as of c-treeACE V9.

Please See
CTRecord::SetFilter(), CTRecord::GetFilter(), CTRecord::IsFiltered()
CTTable::GetCndxIndex

Syntax
void CTTable::GetCndxIndex(NINT indexnbr, CTString& cndexpr)
void CTTable::GetCndxIndex(const CTString& indexname, CTString& cndexpr)

Parameters
- **indexnbr** [in] index number
- **indexname** [in] index name
- **cndexpr** [out] conditional index expression

Description
Retrieves the conditional index expression string, given the index number.

Return
None.

See also
GetCndxIndexByName(), GetCndxIndexLength(), GetCndxIndexLengthByName()
CTTable::GetCndxIndexByName

Syntax
#ifdef PROTOTYPE
CTDBRET ctdbDECL ctdbGetCndxIndexByName(CTHANDLE Handle, pTEXT indexname,
pTEXT buffer, NINT bufferlen)
#else
CTHANDLE Handle;
pTEXT indexname;
pTEXT buffer;
NINT bufferlen;
#endif

Parameters
- **handle** [in]  table handle
- **indexname** [in]  index name
- **buffer** [out]  pointer to buffer to receive the conditional expression
- **bufferlen** [in] size in bytes of buffer

Description
Retrieves the conditional index expression string, given the index name.

Return
Return CTDBRET_OK on success

See also
GetCndxIndex(), GetCndxIndexLength(), GetCndxIndexLengthByName()
CTTable::GetCndxIndexLength

Syntax
#ifdef PROTOTYPE
NINT ctdbDECL ctdbGetCndxIndexLength(CTHANDLE Handle, NINT indexnbr)
#else
NINT ctdbDECL ctdbGetCndxIndexLength(Handle, indexnbr)
CTHANDLE Handle;
NINT indexnbr;
#endif

Parameters
- **handle** [in] table handle
- **indexnbr** [in] index number

Description
Retrieves the length in bytes of the conditional expression string, given the index number.

Return
Return the length in bytes of the conditional expression or -1 on error.

See also
GetCndxIndex(), GetCndxIndexByName(), GetCndxIndexLengthByName()
CTTable::GetCndxIndexLengthByName

Syntax
#ifdef PROTOTYPE
NINT ctdbDECL ctdbGetCndxIndexLengthByName(CTHANDLE Handle, pTEXT indexname)
#else
NINT ctdbDECL ctdbGetCndxIndexLengthByName(Handle, indexname)
#endif
CTHANDLE Handle;
pTEXT indexname;
#endif

Parameters
- `handle` [in] table handle
- `indexname` [in] index name

Description
Retrieves the length in bytes of the conditional expression string, given the index name.

Return
Return the length in bytes of the conditional expression or -1 on error.

See also
GetCndxIndex(), GetCndxIndexByName(), GetCndxIndexLength()
CTTable::GetCreateMode

Syntax

CTCREATE_MODE GetCreateMode( ) const

Parameters

This method has no parameters.

Description

Retrieves the table create mode. The valid values for the table create mode are shown in "Table Create Modes" (page 234).

Return

GetCreateMode() returns the table create mode.

See also

Create(), GetOpenMode()
CTTable::GetDataDefaultExtentSize

Syntax

NINT GetDataDefaultExtentSize( ) const

Parameters

This method has no parameters.

Description

Retrieves the c-tree data file default extent size. The default value, 8192 bytes, may be changed by calling SetDataDefaultExtentSize().

Return

GetDataDefaultExtentSize() returns the default data file extent size.

See also

SetDataDefaultExtentSize(), GetIndexDefaultExtentSize()
CTTable::GetDataExtension

Syntax
CTString GetDataExtension( ) const

Parameters
This method has no parameters.

Description
Gets the data file name extension. If not changed by SetDataExtension(), the data extension is "dat".

Return
GetDataExtension() returns the data file name extension

See also
SetDataExtension(), GetIndexExtension()
**CTTable::GetDatno**

Retrieve the table data file number.

**Declaration**

```cpp
NINT CTTable::GetDatno()
```

**Description**

Retrieves the table data file number from a table object. If an error is detected, `GetDatno()` throws a `CTException`.

**Return**

`GetDatno()` returns the table data file number.

**Example**

```cpp
void DeleteTable(CTSession& hSession, pTEXT tablename)
{
    CTDBRET eRet;
    CTTable hTable(hSession);

    // open the table exclusive
    hTable.Open(tablename, CTOPEN_EXCLUSIVE);

    // delete a file
    eRet = (CTDBRET)DeleteRFile((COUNT)hTable.GetDatno());
    if (eRet != CTDBRET_OK)
        throw CTException(eRet, "DeleteTable", __FILE__, __LINE__);
}
```

**See Also**

`CTBase::SwitchInstance()`, `CTRecord::SwitchContext()`, `CTTable::GetIdxno()`, `CTIndex::GetIdxno()`
CTTable::GetField

Syntax

```cpp
CTField GetField(NINT FieldNumber) const
CTField GetField(const CTString& FieldName) const
```

Parameters

- `FieldNumber` [in] The field number to be retrieved from the table. The first field in the table is field number zero.
- `FieldName` [in] The field name to be retrieved from the table.

Description

Retrieves a field by its number or name. The first field in the table is the field number zero.

Return

`GetField()` returns the field object.

See also

`GetFieldNumber()`, `GetFieldCount()`
CTTable::GetFieldCount

Syntax
VRLENGetFieldCount( ) const

Parameters
This method has no parameters.

Description
Retrieves the number of fields in the table

Return
GetFieldCount() returns the number of fields.

See also
AddField(), DelField(), InsertField(), GetFieldNumber(), GetIndexCount()
CTTable::GetFieldNumber

Syntax
NINT GetFieldNumber(const CTString& FieldName) const

Parameters
- **FieldName** [in] The field name whose field number is to be retrieved from the table.

Description
Retrieves the field number from a table, given its field name.

Return
GetFieldNumber() returns the field number.

See also
GetField(). GetFieldCount()
CTTable::GetFilter

Syntax
void GetFilter (CTString& cndexpr)

Deprecated
Returns CTDBRET_NOTSUPPORTED. c-treeDB Filters are record based as of c-treeACE V9.

Please See
CTRecord::SetFilter(), CTRecord::GetFilter(), CTRecord::IsFiltered()
CTTable::GetFirstPartition

Syntax
LONG CTTable::GetFirstPartition()

Parameters
None

Description
Retrieves the first raw partition number, if the table is a partitioned file.

Return
CTException is thrown if the table is not a partitioned file.

See Also
GetLastPartition()
CTTable::GetGroupid

Syntax
CTString GetGroupid() const

Parameters
This method has no parameters.

Description
Retrieves the table group id

Return
GetGroupid() returns the table group id.

See also
SetGroupid()
CTTable::GetIdxno

Retrieve a table index file number given the index name or the c-treeDB index number.

Declaration
NINT CTTable::GetIdxno(const CTString& IndexName)
NINT CTTable::GetIdxno(NINT index)

Description
Retrieves the index file number given the index name or the c-treeDB index number.

- **IndexName** contains the symbolic name for the index. Please note the difference between the index symbolic name and the index file name.
- **index** is a c-treeDB index number. The first c-treeDB index number is zero.

Return
GetIdxno() returns a table index file number.

Example

```c
// retrieve the first key of first index
TEXT keyval[256];

if (FirstKey(hTable.GetIdxno("IndexName"), keyval)
    printf("FirstKey failed\n");

if (FirstKey(hTable.GetIdxno(0), keyval)
    printf("FirstKey failed\n");
```

See Also
CTBase::SwitchInstance(), CTRecord::SwitchContext(), CTTable::GetDatno(), CTIndex::GetIdxno()
CTTable::GetIndex

Syntax
CTIndex GetIndex(NINT IndexNumber)
CTIndex GetIndex(const CString& name)

Parameters
- **IndexNumber** [in] The Index number to be retrieved.
- **name** [in] The index name to be retrieved.

Description
Retrieves an index object, given its number or name.

Return
GetIndex() returns the Index object.

See also
DelIndex(), AddIndex()
CTTable::GetIndexByUID

Syntax
CTIndex GetIndexByUID(ULONG uid)

Parameters
- **uid** [in] the index unique identifier.

Description
Retrieves the index object given its unique identifier.

Return
GetIndexByUID() returns the Index object.

See also
GetIndex()
CTTable::GetIndexCount

Syntax

VRLEN GetIndexCount( ) const

Parameters

This method has no parameters.

Description

Retrieves the number of table indices

Return

GetIndexCount() returns the number of indices.

See also

GetFieldCount()
CTTable::GetIndexDefaultExtentSize

Syntax
NINT GetIndexDefaultExtentSize( ) const

Parameters
This method has no parameters.

Description
Retrieves the c-tree index file default extent size

Return
GetIndexDefaultExtentSize() returns the default index file extent size. The default value, 8192 bytes, may be changed by SetIndexDefaultExtentSize().

See also
SetIndexDefaultExtentSize(), GetDataDefaultExtentSize()
CTTable::GetIndexExtension

Syntax
CTString GetIndexExtension() const

Parameters
This method has no parameters.

Description
 Gets the index file name extension (usually .idx).

Return
 GetIndexExtension() returns the index file name extension. If not changed by SetIndexExtension(), the index extension is "idx".

See also
SetIndexExtension(), GetDataExtension()
**CTTable::GetIndexFileName**

**Syntax**

```cpp
CTString GetIndexFileName(NINT IndexNumber);
CTString GetIndexFileName(const CTString& IndexName);
```

**Parameters**

These overloaded methods identify the index by number (`IndexNumber`) or by name (`IndexName`).

**Description**

Retrieves the index physical file name.

**Return**

`GetIndexFilename()` returns the index physical file name.

**See also**

`CTTable::SetIndexFileName()`
CTTable::GetLastPartition

Syntax
LONG CTTable::GetLastPartition()

Parameters
None

Description
Retrieves the last raw partition number, if the table is a partitioned file.

Return
CTException is thrown if the table is not a partitioned file.

See Also
GetFirstPartition()
CTTable::GetName

Syntax

CTString GetName( ) const

Parameters

This method has no parameters.

Description

Gets the table name, without the name extension.

Return

GetName() returns the table name.

Example

pTable=pDatabase->GetFirstActive();
printf("First Active table: %s", pTable->GetName().c_str());

See also

getPath(), GetOpenMode(), GetPassword(), GetPermission()
CTTable::GetOpenMode

Syntax
CTOPEN_MODE GetOpenMode( ) const

Parameters
This method has no parameters.

Description
Retrieves the table open mode. The valid values for the table open mode are shown in "Table Open Modes" (page 235).

Return
GetOpenMode() returns the table open mode.

See also
Open(), GetCreateMode()
**CTTable::GetOwner**

**Declaration**

```cpp
CTString CTTable::GetOwner() const
```

**Description**

Retrieves the table owner.

**Return Values**

If the table owner was not previously set by `SetOwner()`, `GetOwner()` returns an empty string. `GetOwner()` returns the table owner or empty string if no table owner is set.

**See Also**

`SetOwner()`
**CTTable::GetPadChar**

**Syntax**

```cpp
void GetPadChar(TEXT &Padchar, TEXT &Dmlchar)
```

**Parameters**

- *Padchar* [out] receive the pad character
- *Dmlchar* [out] receive the field delimiter character

**Description**

Retrieve the table pad and field delimiter characters. These characters are used to pad fixed string fields (*CT_FSTRING*) to allow proper target key formation.

**Return**

None.

**See also**

*SetPadChar()**, *UpdatePadChar()*
CTTable::GetPassword

Syntax

CTString GetPassword( ) const

Parameters

This method has no parameters.

Description

Retrieves the table password

Return

GetPassword() returns the table password

See also

SetPassword(), GetName(), GetPath()
CTTable::GetPath

Syntax
CTString GetPath() const

Parameters
This method has no parameters.

Description
Gets the table path.

Return
GetPath() returns the table path.

See also
GetName()
CTTable::GetPermission

Syntax
LONG GetPermission( ) const

Parameters
This method has no parameters.

Description
Retrieves the table permission mask. The valid values for the table permission mask are shown in "Table Permissions" (page 236).

Return
GetPermission() returns the table permission mask.

See also
SetPermission()
CTTable::GetSegment

Syntax
CTSegment GetSegment(const CTIndex& pIndex, VRLEN SegNumber)

Parameters
- pIndex [in] The index object whose segment is to be retrieved
- SegNumber [in] The segment number to be retrieved

Description
Retrieves an index segment object

Return
GetSegment() returns the segment object

See also
AddSegment(), InsertSegment(), DelSegment()
CTTable::GetStatus

Declaration
ULONG CTTable::GetStatus();

Description
Retrieves the table status. The table status indicates which rebuild action will be taken by an alter table operation.

Return Values
GetStatus() returns CTDB_REBUILD_NONE if the table is not changed or a bit mask of the values below describing the table status.

The possible rebuild status returns are:

<table>
<thead>
<tr>
<th>Rebuild Status Value</th>
<th>Rebuild Status Symbolic Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CTDB_REBUILD_NONE</td>
<td>Nothing to be done, no changes to table</td>
</tr>
<tr>
<td>1</td>
<td>CTDB_REBUILD_DODA</td>
<td>Update the table DODA</td>
</tr>
<tr>
<td>2</td>
<td>CTDB_REBUILD_RESOURCE</td>
<td>Update table FC/DFLD resource</td>
</tr>
<tr>
<td>4</td>
<td>CTDB_REBUILD_INDEX</td>
<td>Add new indices to table</td>
</tr>
<tr>
<td>8</td>
<td>CTDB_REBUILD_ALL</td>
<td>Rebuild all indices</td>
</tr>
<tr>
<td>16</td>
<td>CTDB_REBUILD_FULL</td>
<td>Full table rebuild. A temporary table is created and all data is moved to new table and the indexes are built on the fly.</td>
</tr>
</tbody>
</table>

Example
ULONGLONG status = hTable.GetStatus();

if (status & CTDB_REBUILD_ALL || status & CTDB_REBUILD_FULL)
    hTable.Alter(CTDB_ALTER_FULL)
else if (status)
    hTable.Alter(CTDB_ALTER_NORMAL);

See Also
CTIndex::GetStatus(), CTSegment::GetStatus(), CTTable::Alter()
CTTable::GetTableKSeg

Retrieves the current table wide extended key segment definition.

Declaration

```cpp
void CTTable::GetTableKSeg(pctKSEGDEF pKSeg);
```

Description

`CTTable::GetTableKSeg()` retrieves the current table wide extended key segment definition. 
`pKSeg` is a pointer to an extended key segment definition structure which will receive the definition.

Return

`void`

Example

```cpp
ctkSEGDEF kseg;
hTable.GetTableKSeg(&ksge);
```

See Also

`CTIndex::GetIndexKSeg()`, `CTIndex::SetIndexKSeg()`, `CTSegment::GetSegmentKSeg()`, `CTSegment::SetKSegDefaults()`, `CTSegment::SetSegmentKSeg()`, `CTTable::SetTableKSeg()`
CTTable::HasDelField

Determines if a table has a delete $DEL$ field.

**Declaration**

```cpp
bool CTTable::HasDelField()
```

**Description**

`CTTable::HasDelField()` indicates if a table associated with the current handle has a delete field.

**Return**

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO</td>
<td>Table does not have a $DEL$ field.</td>
</tr>
<tr>
<td>1</td>
<td>YES</td>
<td>Table has a $DEL$ field.</td>
</tr>
</tbody>
</table>

**Example**

```cpp
if myTable->HasDelField()
    printf("My Table has a Delete Field in it\n");
```
CTTable::HasNullFieldSupport

Syntax
CTBOOL HasNullFieldSupport( )

Parameters
This method has no parameters.

Description
Indicates if a table has null field support.

Return
HasNullFieldSupport() returns YES if the table has null field support, NO otherwise.
CTTable::HasRecbyt

Syntax
CTBOOL HasRecbyt( )

Parameters
This method has no parameters.

Description
Indicates if a table has support to a recbyt index or not. By default, when a table is created it has support to this index enabled.

Return
HasRecbyt() returns YES if the table has support to the recbyt index, NO otherwise.

See also
Create()
CTTable::HasRowid

Syntax
CTBOOL HasRowid( )

Parameters
This method has no parameters.

Description
Indicates if a table has support to a rowid index or not. By default, when a table is created it has support to this index enabled. A rowid index is an automatic sequence number.

Return
HasRecbyt() returns YES if the table has support to the rowid index, NO otherwise.

See also
Create(), Record::FindRowid(), Record::GetRowid()
CTTable::InsertField

Syntax

CTField InsertField(const NINT BeforeField, const CTString& Name, 
                    CTDBTYPE Type, VRLEN Length)
CTField InsertField(const CTString& BeforeField, const CTString& Name, 
                    CTDBTYPE Type, VRLEN Length)

Parameters

- **BeforeField** [in] Insert the new field before this field number, if the prototype `InsertField(const NINT BeforeField, const CTString& Name, CTDBTYPE Type, VRLEN Length)` is used. Insert the new field before this field name, if the prototype `InsertField(const CTString& BeforeField, const CTString& Name, CTDBTYPE Type, VRLEN Length)` is used.
- **Name** [in] The field name to insert in the table
- **Type** [in] the field type. Valid field type values are shown in "Field Types" (page 227).
- **Length** [in] the field length

Description

Inserts a new field into the table, in the specified position.

Return

`InsertField()` returns a field object on success or NULL on failure.

See also

`AddField()`, `DelField()`, `GetFieldNumber()`
**CTTable::InsertSegment**

### Syntax

```cpp
CTSegment InsertSegment(const CTIndex& pIndex, NINT BeforeSegment, const CTFIELD& Field, CTSEG_MODE SegMode)
CTSegment InsertSegment(const CTIndex& pIndex, NINT BeforeSegment, const CString& FieldName, CTSEG_MODE SegMode)
CTSegment InsertSegment(const CTIndex& pIndex, NINT BeforeSegment, NINT FieldNumber, CTSEG_MODE SegMode)
CTSegment InsertSegment(const CTIndex& pIndex, NINT BeforeSegment, NINT offset, NINT length, CTSEG_MODE SegMode)
CTSegment InsertSegment(NINT IndexNbr, NINT BeforeSegment, NINT FieldNumber, CTSEG_MODE SegMode)
CTSegment InsertSegment(NINT IndexNbr, NINT BeforeSegment, const CString& FieldName, CTSEG_MODE SegMode)
CTSegment InsertSegment(const CString& IndexName, NINT BeforeSegment, const CString& FieldName, CTSEG_MODE SegMode)
CTSegment InsertSegment(const CString& IndexName, NINT BeforeSegment, NINT FieldNumber, CTSEG_MODE SegMode)
CTSegment InsertSegment(const CString& IndexName, NINT BeforeSegment, const CTFIELD& Field, CTSEG_MODE SegMode)
CTSegment InsertSegment(NINT IndexNbr, NINT BeforeSegment, const CTFIELD& Field, CTSEG_MODE SegMode)
CTSegment InsertSegment(const CString& IndexName, NINT BeforeSegment, const CTFIELD& Field, CTSEG_MODE SegMode)
```

### Parameters

- `pIndex [in]` A pointer to the index object.
- `BeforeSegment [in]` Insert the new segment before this segment.
- `Field [in]` A pointer to the field object that composes part of the index segment
- `SegMode [in]` The segment mode. The valid values to the segment mode are shown in "Segment Modes" (page 92).
- `FieldName [in]` Field name to compose the index segment
- `FieldNumber [in]` Field number to compose the index segment
- `offset [in]` Absolute byte offset of the segment
- `Length [in]` Segment length in bytes
- `IndexNbr [in]` Index number
- `IndexName [in]` Index name
- `Field [in]` Field segment to be inserted

### Description

Inserts a new segment in an index.

### Return

`InsertSegment()` returns a segment object.

### See also

`AddSegment()`, `DelSegment()`
CTTable::IsActive

Syntax
CTBOOL IsActive() const

Parameters
This method has no parameters.

Description
Indicates if a table is open or closed.

Return
IsActive() returns YES if the table is open and NO otherwise.

Example
if (!pTable->IsActive())
    pTable->Open(tb_name, "CTOPEN_NORMAL");

See also
Open()
CTTable::IsFilteredRecord

Syntax
CTBOOL IsFilteredRecord( )

Deprecated
Returns CTDBRET_NOTSUPPORTED. c-treeDB Filters are record based as of c-treeACE V9.

Please See
CTRecord::SetFilter(), CTRecord::GetFilter(), CTRecord::IsFiltered()
CTTable::MoveField

Syntax
void MoveField(const NINT FieldNumber, NINT newNumber)
void MoveField(const CTString& FieldName, NINT newNumber)

Parameters
- FieldNumber [in] The field number to be moved, if the prototype MoveField(const NINT FieldNumber, NINT newNumber) is used.
- newNumber [in] The new field number in the table.
- FieldName [in] The field name to be moved, if the prototype MoveField(CTString& FieldName, NINT newNumber) is used.

Description
Moves the field from one position to another position in the same table.

Return
None.

See also
AddField(), InsertField(), DelField()
CTTable::MoveSegment

Moves a key segment to a location.

Declaration

void CTTable::MoveSegment(CTIndex hIndex, NINT segNumber, NINT newIndex);
void CTTable::MoveSegment(NINT indexNumber, NINT segNumber, NINT newIndex);
void CTTAble::MoveSegment(const CTString& indexName, NINT segNumber,
                          NINT newIndex);

Description

CTTable::MoveSegment() moves a key segment to a location indicated by newIndex.

- hIndex is an index object.
- indexNumber is a relative number for a index object in a table definition.
- indexName is the unique name for the index.
- newIndex indicates the relative position were the key segment should be moved to.

Return
void

Example

// move the last segment to first
CTIndex hIndex = hTable.GetIndex(0);
NINT count = hIndex.GetSegmentCount();
if (count > 0)
    hTable.MoveSegment(hIndex, (count - 1), 0);

See Also

CTIndex::MoveSegment(), CTSegment::MoveSegment()
**CTTable::Open**

**Syntax**

```cpp
void Open(const CTString& Name, const CTOPEN_MODE OpenMode)
```

**Parameters**

- *Name* [in] The table name
- *OpenMode* [in] The open mode for the table. The valid values for the table open mode are shown in "Table Open Modes" (page 235).

**Description**

Opens an existing table.

**Return**

None.

**See also**

GetOpenMode(), Create(), Close()
CTTable::PartAdminByKey

Performs an administrative partition operation on the partition specified by key value.

**Declaration**

COUNT PartAdminByKey(pVOID key, CTPART_MODE_E partmode);

**Description**

PartAdminByKey() manages the partitions for a table.

- **keyval** [in] - pointer to a partition key value that has been transformed
- **partmode** [in] - partition operation mode. `partmode` is one of:
  
  - `CTPART_PURGE` - delete a partition
  - `CTPART_ADD` - add a partition
  - `CTPART_ARCHIVE` - archive a partition
  - `CTPART_BASE` - modify the lower limit partition number value
  - `CTPART_ACTIVAT_E` - activate an archived partition
  - `CTPART_STATUS` - return the partition status in `partstatus`

**Return Values**

`CTDBRET_NOERROR` is returned if no error.

If `CTPART_STATUS` is passed as a partition mode, then one of the following `partstatus` values is returned:

<table>
<thead>
<tr>
<th><code>partstatus</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmSTATUSnone</td>
<td>partition member does not exist</td>
</tr>
<tr>
<td>pmSTATUSexst</td>
<td>partition member is active</td>
</tr>
<tr>
<td>pmSTATOpn</td>
<td>partition member is active and currently open</td>
</tr>
<tr>
<td>pmSTATAsrchv</td>
<td>partition member is archived</td>
</tr>
<tr>
<td>pmSTATSpurg</td>
<td>partition member was purged</td>
</tr>
<tr>
<td>pmSTATSparc</td>
<td>partition member is pending archive</td>
</tr>
<tr>
<td>pmSTATSpn</td>
<td>partition member is pending purge</td>
</tr>
</tbody>
</table>

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

**See Also**

CTTable::PartAdminByName(), CTTable::PartAdminByNumber()
CTTable::PartAdminByName

Performs an administrative partition operation on the partition specified by partition name.

Declaration
COUNT PartAdminByName(pTEXT partname, CTPART_MODE_E partmode);

Description
PartAdminByName() manages the partitions for a table.

- **partname** [in] - partition name.
- **partmode** [in] - partition operation mode. partmode is one of:
  - CTPART_PURGE - delete a partition
  - CTPART_ADD - add a partition
  - CTPART_ARCHIVE - archive a partition
  - CTPART_BASE - modify the lower limit partition number value
  - CTPART_ACTIVATE - activate an archived partition
  - CTPART_STATUS - return the partition status in partstatus

Return Values
CTDBRET_NOERROR is returned if no error.

If CTPART_STATUS is passed as a partition mode, then one of the following partstatus values is returned:

| pmSTATUSnone | 0 | partition member does not exist |
| pmSTATUSexst | 1 | partition member is active |
| pmSTATUSopnd | 2 | partition member is active and currently open |
| pmSTATUSarhv | 3 | partition member is archived |
| pmSTATUSpurg | 4 | partition member was purged |
| pmSTATUSparc | 19 | partition member is pending archive |
| pmSTATUSppnd | 20 | partition member is pending purge |

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

See Also
CTTable::PartAdminByKey(), CTTable::PartAdminByNumber()
CTTable::PartAdminByNumber
Performs an administrative partition operation on the partition specified by partition number.

Declaration
COUNT PartAdminByNumber(LONG partno, CTPART_MODE_E partmode);

Description
PartAdminByKey() manages the partitions for a table.
- partno [in] - partition number
- partmode [in] - partition operation mode. partmode is one of:
  - CTPART_PURGE delete a partition
  - CTPART_ADD add a partition
  - CTPART_ARCHIVE archive a partition
  - CTPART_BASE modify the lower limit partition number value
  - CTPART_ACTIVATE activate an archived partition
  - CTPART_STATUS return the partition status in partstatus

Return Values
CTDBRET_NOERROR is returned if no error.
If CTPART_STATUS is passed as a partition mode, then one of the following partstatus values is returned:

<table>
<thead>
<tr>
<th>pmSTATUSnone</th>
<th>0</th>
<th>partition member does not exist</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmSTATUSexst</td>
<td>1</td>
<td>partition member is active</td>
</tr>
<tr>
<td>pmSTATUSopnd</td>
<td>2</td>
<td>partition member is active and currently open</td>
</tr>
<tr>
<td>pmSTATUSarhv</td>
<td>3</td>
<td>partition member is archived</td>
</tr>
<tr>
<td>pmSTATUSpurg</td>
<td>4</td>
<td>partition member was purged</td>
</tr>
<tr>
<td>pmSTATUSparc</td>
<td>19</td>
<td>partition member is pending archive</td>
</tr>
<tr>
<td>pmSTATUSppnd</td>
<td>20</td>
<td>partition member is pending purge</td>
</tr>
</tbody>
</table>

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

See Also
CTTable::PartAdminByKey(). CTTable::PartAdminByName()
CTTable::Rebuild

Declaration

CTTable::Rebuild(CTREBUILD_MODE mode);

Description

- See Also Handle [IN] Table handle
- mode [IN] The following modes are available:

<table>
<thead>
<tr>
<th>CTREBUILD Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTREBUILD_NONE</td>
<td>The normal rebuild mode.</td>
</tr>
<tr>
<td>CTREBUILD_PURGEDUP</td>
<td>Purge duplicate records during rebuild.</td>
</tr>
<tr>
<td>CTREBUILD_UPDATEIFIL</td>
<td>Update the IFIL structure in the table.</td>
</tr>
<tr>
<td>CTREBUILD_DATAONLY</td>
<td>Rebuild only the datafile.</td>
</tr>
</tbody>
</table>

Note: Exercise care when using CTREBUILD_DATAONLY and CTREBUILD_UPDATEIFIL modes together as the index files will be removed from the table IFIL definition even if the index files still exist in the file system. This can cause later problems if Alter() is called to recreate the removed indices.

Rebuild() calls the c-treeACE CTRBLIFILX() function to rebuild a c-tree table. When used in conjunction with the open modes CTOPEN_CORRUPT and CTOPEN_DATAONLY, the Rebuild() function can be used as a direct replacement for the c-tree Plus ctrbldif rebuild utility.

The following steps are performed by c-treeDB during a table rebuild process:

1. If a transaction is active, and the table being rebuilt was created with CTCREATE_TRNLOG or CTCREATE_PREIMG, the transaction is committed before the table is rebuilt, and the transaction is restarted after the table rebuild process is completed.
2. The update corrupt flag, updflg, of the header of the data file is cleared.
3. The internal delete stack chain of fixed-length data files or the internal delete management index of variable length data files are rebuilt.
4. The existing index files are removed and new index files are rebuilt over the existing files, optimized for both size and speed.

You must open the table before Rebuild() is executed. It is recommended that the table be opened with CTOPEN_EXCLUSIVE mode. If the table is corrupt, you will need to open the table with the CTOPEN_CORRUPT mode and then call Rebuild() to rebuild the data and index files. If there are missing or corrupt index files, open the table with CTOPEN_DATAONLY mode and Rebuild() will reconstruct the missing index files.

There may be situations when you need to invoke this function to rebuild only the data file. After the data file rebuild is successful, you may need to call Rebuild() again to rebuild the index files.

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO_ERROR</td>
<td>Successful operation.</td>
</tr>
<tr>
<td>650</td>
<td>DUPJ_ERR</td>
<td>Duplicate keys purged and logged.</td>
</tr>
</tbody>
</table>
See c-tree Plus Error Codes (http://docs.faircom.com/doc/ctreeplus/28320.htm) for a complete listing of valid c-tree Plus error values.

See Also

Alter()
CTTable::Remove

Removes any c-treeDB table.

Declaration
void CTTable::Remove();

Description
The CTTable::Remove() function allows any table to be deleted, including tables that are not members of a database.

The CTTable::Remove() function deletes a c-tree data file and the associated index files from disk. If the table was opened under a database handle, the table is closed and CTDATABASE::DeleteTable() is called. If the handle is not active, the table is opened exclusive and then deleted. If the table handle is not active, you must set the path, file extension and password for the handle before calling the RemoveTable() method.

A CTEXCEPTION is thrown when an error occurs. Use CTEXCEPTION::GetErrorCode() and CTEXCEPTION::GetErrorMsg() to retrieve the error code and descriptive message.

Return
void

Example

/* delete a c-treeDB Table */
try {
    MyTable.Remove();
}
catch(CTException &E) {
    printf("MyTable.Remove error: (%d) - %s\n",
           E.GetErrorCode(),
           E.GetErrorMsg());
}

See Also
CTDATABASE::DeleteTable()
**CTTable::ResetAll**

**Syntax**

```cpp
void ResetAll();
```

**Parameters**

This method has no parameters.

**Description**

Resets all active record buffers associated with the table.

**Return**

None.
CTTable::SetDataDefaultExtentSize

Syntax
void SetDataDefaultExtentSize(const NINT size)

Parameters
- **size** [in] The default data extent size. The data extent default value is 8192 initially.

Description
Sets the c-tree data file default extent size. Each time the data file needs to be extended, it will use this value to extent the file. If not changed by `SetDataDefaultExtentSize()`, this value is 8192 bytes.

Return
None.

See also
`GetDataDefaultExtentSize()`, `SetIndexDefaultExtentSize()`
CTTable::SetDataExtension

Syntax
void SetDataExtension(const CTString& Ext)

Parameters
- Ext [in] The table extension, usually "dat" (default value).

Description
Sets the table data file extension. If not changed by SetDataExtension(), the data extension is "dat".

Return
None.

See also
GetDataExtension(), SetIndexExtension()
CTTable::SetGroupid

Syntax
void SetGroupid(const CTString& Groupid)

Parameters

Description
Sets the table group id.

Return
None.

See also
GetGroupid()
CTTable::SetIndexDefaultExtentSize

Syntax
void SetIndexDefaultExtentSize(const NINT size)

Parameters
- size [in] The default index extent size. The index extent default value is 8192 initially.

Description
Sets the c-tree index file default extent size. Each time the index file needs to be extended, it will use this value to extent the file. If not changed by SetIndexDefaultExtentSize(), this value is 8192 bytes.

Return
None.

See also
GetIndexDefaultExtentSize(), SetDataDefaultExtentSize()
CTTable::SetIndexExtension

Syntax
void SetIndexExtension(const CString& Ext)

Parameters
- Ext [in] The table index file extension, usually "idx" (default value).

Description
Sets the table index file extension. If not changed by SetIndexExtension(), the index extension is "idx".

Return
None.

See also
GetIndexExtension(), SetDataExtension()
CTTable::SetIndexFileName

Syntax
void CTTable::SetIndexFileName(NINT IndexNumber, const CTString* path,
                                const CTString* filename);
void CTTable::SetIndexFileName(const CTString& IndexName, const CTString* path,
                                const CTString* filename);

Parameters
These overloaded methods identify the index by number (IndexNumber) or by name (IndexName).

- path specifies the directory location of the index file. A NULL value for path indicates that the index file is to be located in the same directory as the data file.
- filename specifies the name of the index file. If filename is NULL, the index file name is cleared.

An application can change the current index file extension by calling the CTTable::SetIndexExtension() method.

Description
Sets the physical file name for the index.

Return
None.

See also
CTTable::GetIndexFileName(), CTTable::SetIndexExtension(),
CTIndex::GetIndexFileName()
CTTable::SetPadChar

Syntax
void SetPadChar(NINT Padchar, NINT Dmlchar)

Parameters
- Padchar [in] pad character
- Dmlchar [in] field delimiter character

Description
Set the table pad and field delimiter characters in the table object. These characters are used to pad fixed string fields (CT_FSTRING) to allow proper target key formation.

Return
None.

See also
GetPadChar(), UpdatePadChar()
CTTable::SetPassword

Syntax
void SetPassword(const CTString& Password)

Parameters
- **Password** [in] The new table password.

Description
Sets the table password

Return
None.

See also
GetPassword()
CTTable::SetPath

Syntax

void SetPath(const CTreeDB::CPath& Path)

Parameters

- Path [in] the new table path.

Description

Sets the table path.

Return

None.

See also

GetPath()
CTTable::SetPermission

Syntax
void SetPermission(const LONG Permmask)

Parameters
- **Permmask** [in] The table permission mask. The valid values for the table permission mask are shown in "Table Permissions" (page 236).

Description
Sets the table permission mask.

Return
None.

See also
GetPermission()
CTTable::SetTableKSeg
Establishes a table wide extended key segment definition.

Declaration
void CTTable::SetTableKSeg(pctKSEGDEF pKSeg);

Description
CTTable::SetTableKSeg() establishes a table wide extended key segment definition. pKSeg is a pointer to an extended key segment definition structure with the extended key definition.

Return
void

Example
ctKSEGDEF kseg;

kseg.kseg_ssiz = ctKSEG_SSIZ_COMPUTED;
kseg.kseg_type = ctKSEG_TYPE_UNICODE;
kseg.kseg_styp = ctKSEG_STYP_UTF16;
kseg.kseg_comp = ctKSEG_COMPU_S_DEFAULT | ctKSEG_COMPU_N_NONE;
kseg.kseg_desc = "en_US";
hTable.SetTableKSeg(&kseg);

See Also
CTIndex::GetIndexKSeg(), CTIndex::SetIndexKSeg(), CTSegment::GetSegmentKSeg(), CTSegment::SetKSegDefaults(), CTSegment::SetSegmentKSeg(), CTTable::GetTableKSeg()
CTTable::SetOwner

Declaration
void CTTable::SetOwner(const CString& owner)

Description
Sets the table owner. You should set the table owner before the table is created to allow the proper c-tree security setting to take place.

Return Values
None.

See Also
CTTable::GetOwner()
CTTable::SystemFilterOff

Deletes a permanent system wide record filter.

Declaration
void CTTable::SystemFilterOff(CTSYSFILTER mode);

Description
Deletes a permanent system wide record filter. The table must be opened exclusive and the user must have file definition permission for the table.

mode specifies one of the following:
- CTSYSFILTER_READ - Indicates you are deleting only a system wide read record filter.
- CTSYSFILTER_WRITE - Indicates you are deleting only a system wide write record filter
- CTSYSFILTER_READ | CTSYSFILTER_WRITE - Indicates you are deleting both a system wide read record filter and a system wide write record filter.

Return
On error CTTable::SystemFilterOff() throws a CException exception object.

See Also
CTTable::SystemFilterOn()
CTTable::SystemFilterOn

Establishes a permanent system wide data record filter.

Declaration

```cpp
void CTTable::SystemFilterOn(CTSYSFILTER mode);
```

Description

Establishes a permanent system wide, i.e. the filter applies to all users, read and/or write record filter. Depending on server file security setting, the table must be opened exclusive and the user must have file definition permission for the table. Once a read or a write filter is established, it can only be deleted by calling method CTTable::SystemFilterOff().

A table may have at most one read and one write system wide filter. A write filter will be called when data records are added, updated or deleted.

`mode` specifies one of the following:

- `CTSYSFILTER_READ` - Indicates you are deleting only a system wide read record filter.
- `CTSYSFILTER_WRITE` - Indicates you are deleting only a system wide write record filter
- `CTSYSFILTER_READ | CTSYSFILTER_WRITE` - Indicates you are deleting both a system wide read record filter and a system wide write record filter.

System wide filters must be callback filters. The actual callback evaluation takes place in a new callback function `ctfiltercb_rowl()` located in module `ctclbk.c`.

There different levels of security settings when users modify data file definition resources such as `IFIL` and `DODA`. The c-tree Server can be configured for three different levels of data file resource security:

**FILEDEF_SECURITY_LEVEL LOW:**

Lowest security setting. There is no protection as any user may add or delete data file definition resources. This setting may be used to keep the c-tree Server data compatible with legacy applications.

**FILEDEF_SECURITY_LEVEL MEDIUM:**

Default security setting. Any user may add or delete data file definition resources, but the file must be opened exclusive. This default setting may be enough to keep the c-tree Server data compatible with most legacy applications.

**FILEDEF_SECURITY_LEVEL HIGH:**

Highest security setting. A user must have file definition permission before a definition resource is added or deleted. The file must be opened exclusive. This setting is appropriate for applications that require the highest level of security and may cause compatibility problems with existing legacy applications.

Return

On error CTTable::SystemFilterOn() throws a CException exception object.

Example
try
{
    // open an existing table
    hTable.Open("table_name", CTOPEN_EXCLUSIVE);
    // establish a new system wide read-write filter
    hTable.SystemFilterOn(CTSYSFILTER_READ | CTSYSFILTER_WRITE);
    // close the table
    hTable.Close();
}
catch (CTException &err)
{
    printf("Setting system filter failed with error %d\n", err.GetErrorCode());
}

See Also

CTTable::SystemFilterOff()
CTTable::UpdateCreateMode

Syntax

void CTTable::UpdateCreateMode(CTCREATE_MODE mode)

Parameters

- **mode** [in] Represents the new table create mode. It must be perfectly formed, as it will replace the current table create mode. Use the method GetCreateMode() to retrieve the current create mode and apply the changes on a fully qualified create mode. Update only the following create table modes:
  - CTCREATE_PREIMG
  - CTCREATE_TRNLOG
  - CTCREATE_WRITETHRU
  - CTCREATE_CHECKLOCK
  - CTCREATE_CHECKREAD
  - CTCREATE_HUGEFILE

Description

Update the table create mode. This method changed critical file mode attributes such as the level of transaction control.

No check is made to determine if the mode change will damage data. No check is made if the new mode is valid.

**Note:** Use this method with caution as data may be lost. For instance, changing a data file from transaction processing to no transaction processing make automatic recovery unavailable.

Return

None.

See also

Create()
CTTable::UpdatePadChar

Syntax
void UpdatePadChar(NINT Padchar, NINT Dmlchar, CTBOOL rebuild)

Parameters
- Padchar [in] pad character
- Dmlchar [in] field delimiter character
- rebuild [in] - NOT YET IMPLEMENTED. Indicates if the table should be rebuilt. If rebuild is set to YES, every record is read and the fixed string fields (CT_FSTRING) are padded according to new padding strategy.

Description
Update the table pad and delimiter character resource. The table must be opened exclusive to allow update of the resource.

Return
None.

See also
GetPadChar(), SetPadChar()
CTTable::UnlockTable

Syntax

void UnlockTable( )

Parameters

None.

Description

Unlocks all record locks from the table, if they were obtained with a call to CTRecord::LockRecord(). To free a specific record lock obtained with LockRecord(), use CTTable::UnlockTable().

Return

None.

See also

CTRecord::LockRecord(), CTRecord::UnlockRecord()
4.20 CTTime Class

class CTTime

Description
The CTTime class represents Time objects.

See also
CTTimeTime, CTTime

Preconditions
This is one of the basic objects from the c-tree database layer.

CTTime Operators
- operator == : Assignment operator
- operator <, <=, >, >=, ==, != : Comparison operators
CTTime::operator =

Syntax
CTTime& operator = (const CTTime& Time)
CTTime& operator = (CTTIME Time)

Parameters
- Time [in] The CTTime object or Time to be assigned to the new CTTime object

Description
These overloaded operators assign a Time or CTTime object to a CTTime object

Return
The new CTTime object
CTTime::operator <, <=, >, >=, ==, !=

Syntax
CTBOOL operator < (const CTTime& Time)
CTBOOL operator <= (const CTTime& Time)
CTBOOL operator > (const CTTime& Time)
CTBOOL operator >= (const CTTime& Time)
CTBOOL operator == (const CTTime& Time)
CTBOOL operator != (const CTTime& Time)

Parameters
- Time [in] The Time to be compared to this CTTime object

Description
These overloaded operators make comparisons between the CTTime object and Time.

Return
The operators return YES or NO, depending on the result of the comparison.
CTTime Methods

Constructor / Destructor
- **CTTime()**: Creates a *CTTime* object
- **~CTTime()**: Destroys a *CTTime* Object and resets all the dependent objects

Time Handling
- **GetTime()**: Retrieves the Time.
- **SetTime()**: Sets the Time to the *CTTime* object.
- **Pack()**: Packs an hour, minute and second value into a *CTTIME* object.
- **Unpack()**: Unpacks a *CTTIME* object into hour, minute and second values
- **Hour()**: Retrieves the hour component from a *CTTIME* object
- **Minute()**: Retrieves the minute component from a *CTTIME* object
- **Second()**: Retrieves the second component from a *CTTIME* object
- **TimeToString()**: Converts a packed *CTTIME* object to a *CTString* object
- **StringToTime()**: Converts a *CTString* object to a Time object
- **CurrentTime()**: Retrieves the current system Time.
CTTable::Attach

Attaches a c-tree Plus ISAM `datno` object to a c-treeDB table object.

**Declaration**

```cpp
void CTTable::Attach(NINT datno);
```

**Description**

- `datno` is data file number opened with one of the c-tree ISAM open functions.

**CTTable::Attach()** attaches a c-tree Plus ISAM `datno` object to a c-treeDB table object. This function is useful if you have opened a data and index file using one of c-tree’s ISAM open functions and need to attach it to a table handle to use some of the advanced c-treeDB features such as alter table or the record handler.

**Return**

void

**Example**

```cpp
CTSession hSession(CTSESSION_CTREE);
CTable hTable(hSession);
CTRecord hRecord(hTable);
NINT datno, count = 0;

// logon to c-tree
hSession.Logon(SERVER, USER, PASSWD);

// open the file using c-tree ISAM
datno = (NINT)OPNRFILX((COUNT)-1, "test309.dat", (COUNT)0, NULL);

// attach to table
hTable.Attach(datno);

// read the records
if (hRecord.First())
    do
        { count++;
        } while (hRecord.Next());

// cleanup
hTable.Detach();
hSession.Logout();
```

**See Also**

CTTable::AttachXtd(), CTTable::Detach()
CTTable::AttachXtd

Attaches a c-tree Plus ISAM `datno` object to a c-treeDB table handle allowing a DODA and IFIL to be specified.

**Declaration**

```cpp
void CTTable::AttachXtd(NINT datno, NINT nbrfields, pDATOBJ dodaptr, pIFIL ifilptr);
```

**Description**

- `datno` is the data file number.
- `nbrfields` is the number of fields described by `dodaptr`. If `nbrfields` is zero, it is assumed that the DODA is not supplied and a DODA resource will be read from the table itself.
- `dodaptr` pointer to DODA entries for each field in table. If `dodaptr` is NULL, it is assumed that the DODA is not supplied and a DODA resource will be read from the table itself.
- `ifilptr` is a pointer to IFIL structure describing indices of table. If `ifilptr` is NULL it is assumed that an IFIL is not supplied and an IFIL resource will be read from the table itself.

`CTTable::AttachXtd()` attaches a c-tree Plus ISAM `datno` object to a c-treeDB table object. This function is useful if you have opened a data and index file using one of c-tree’s ISAM open functions and need to attach it to a table handle to use some of the advanced c-treeDB features such as alter table or the record handler. This extended version allows the user to specify the DODA and IFIL for the table, enabling tables without DODA and/or IFIL to be attached to c-treeDB.

**Return**

void

**Example**

```cpp
/* DODA */
static DATOBJ doda[] =
{
    {"f1", (pTEXT)0, CT_INT4, 4},
    {"f2", (pTEXT)4, CT_FSTRING, 10}
};
/* IFIL */
static ISEG iseg = {0, 4, 12};
static IIDX iidx = {4, 0, 0, 0, 1, &iseg, "i311x1", NULL, NULL, NULL};
static IFIL ifil = {"test310", -1, 14, 0, 0, 1, 0, &iidx, "f1", "f2", 0};

CTSession hSession(CTSESSION_CTREE);
CTTable hTable(hSession);
CTRecord hRecord(hTable);
NINT datno, count = 0;

// logon to c-tree
hSession.Logon(SERVER, USER, PASSWD);
// open the file using c-tree ISAM
datno = (NINT)OPNRFILX((COUNT) -1, "test309.dat", (COUNT)0, NULL);
// attach to table
hTable.AttachXtd(datno, 2, doda, &ifil);
// read the records
```
if (hRecord.First())
   do
   {
      count++;
   }
   while (hRecord.Next());
// cleanup
hTable.Detach();
hSession.Logout();
CTTable::ClearAllFieldDefaultValue

Clears the default value associated with all fields in a table.

Declaration

void CTTable::ClearAllFieldDefaultValue();

Description

Clears the default value associated with all fields in a table. The default date and time types for each field are also reset to the default values of CTDATE_MDCY and CTIME_HMS respectively.

Return

void

Example

// clear all field default value
try
{
    hTable.ClearAllFieldDefaultValue();
}
catch (CTException &err)
{
    printf("ClearAllFieldDefaultValue failed\n");
}

See Also

SetFieldDefaultValue(), GetFieldDefaultValue(), IsFieldDefaultValueSet(), ClearFieldDefaultValue(), SetFieldDefaultDateTimeType(), GetFieldDefaultDateType(), GetFieldDefaultTimeType()
CTTime::CTTime

Syntax

CTTime ( )
CTTime(NINT hour, NINT minute, NINT second)
CTTime(CTTIME Time)
CTTime(const CTTime& Time)

Parameters

- **hour** [in] The hour to set to the CTTime object.
- **minute** [in] The minute to set to the CTTime object.
- **second** [in] The second to set to the CTTime object.
- **Time** [in] The Time to set to the CTTime object

Description

This is the constructor for the CTTime object.

See also

~CTTime()
CTTime::~CTTime

Syntax
~CTTime

Parameters
This destructor has no parameters.

Description
This is the destructor for the CTTime object.

See also
CTTime()
CTTime::CurrentTime

**Syntax**

CTTime CurrentTime( )

**Parameters**

This method has no parameters.

**Description**

Retrieves the current system time

**Return**

*CurrentTime()* returns a *CTTime* object with the system current *Time*.

**See also**

*GetTime()*
CTTable::Detach

Detaches a c-treeDB table object from a c-tree data and index files.

Declaration
void CTTable::Detach();

Description
CTTable::Detach() detaches a c-treeDB table object from a c-tree data and index files. The table is not closed but the c-treeDB table handle resources are released and the handle re-initialized.

Return
void

Example

CTSession hSession(CTSESSION_CTREE);
CTTable hTable(hSession);
CTRecord hRecord(hTable);
NINT datno, count = 0;

// logon to c-tree
hSession.Logon(SERVER, USER, PASSWD);

// open the file using c-tree ISAM
datno = (NINT)OPNRFILX((COUNT) -1, "test309.dat", (COUNT)0, NULL);

// attach to table
hTable.Attach(datno);

// read the records
if (hRecord.First())
do{
    count++;
} while (hRecord.Next());

// cleanup
hTable.Detach();
hSession.Logout();

See Also
CTTable::Attach(), CTTable::AttachXtd()
CTTable::GetDatno

Retrieve the table data file number.

Declaration
NINT CTTable::GetDatno()

Description
Retrieves the table data file number from a table object. If an error is detected, GetDatno() throw a CTException.

Return
GetDatno() returns the table data file number.

Example

```c
void DeleteTable(CTSession& hSession, pTEXT tablename)
{
    CTDBRET eRet;
    CTTable hTable(hSession);

    // open the table exclusive
    hTable.Open(tablename, CTOPEN_EXCLUSIVE);

    // delete a file
    eRet = (CTDBRET)DeleteRFile((COUNT)hTable.GetDatno());
    if (eRet != CTDBRET_OK)
        throw CTException(eRet, "DeleteTable", __FILE__, __LINE__);
}
```

See Also
CTBase::SwitchInstance(), CTRecord::SwitchContext(), CTTable::GetIdxno(), CTIndex::GetIdxno()
CTTable::GetIdxno

Retrieve a table index file number given the index name or the c-treeDB index number.

Declaration

NINT CTTable::GetIdxno(const CTString& IndexName)
NINT CTTable::GetIdxno(NINT index)

Description

Retrieves the index file number given the index name or the c-treeDB index number.

- *IndexName* contains the symbolic name for the index. Please note the difference between the index symbolic name and the index file name.
- *index* is a c-treeDB index number. The first c-treeDB index number is zero.

Return

*GetIdxno()* returns a table index file number.

Example

```c
// retrieve the first key of first index
TEXT keyval[256];

if (FirstKey(hTable.GetIdxno("IndexName"), keyval)
    printf("FirstKey failed\n");

if (FirstKey(hTable.GetIdxno(0), keyval)
    printf("FirstKey failed\n");
```

See Also

*CTBase::SwitchInstance*, *CTRecord::SwitchContext*, *CTTable::GetDatno*, *CTIndex::GetIdxno*
CTTable::GetTableKSeg

Retrieves the current table wide extended key segment definition.

Declaration

void CTTable::GetTableKSeg(pctKSEGDEF pKSeg);

Description

CTTable::GetTableKSeg() retrieves the current table wide extended key segment definition. 

pKSeg is a pointer to an extended key segment definition structure which will receive the definition.

Return

void

Example

ctKSEGDEF kseg;
hTable.GetTableKSeg(&ksge);

See Also

CTIndex::GetIndexKSeg(), CTIndex::SetIndexKSeg(), CTSegment::GetSegmentKSeg(), CTSegment::SetKSegDefaults(), CTSegment::SetSegmentKSeg(), CTTable::SetTableKSeg()
CTTime::GetTime

Syntax
CTTIME GetTime( ) const

Parameters
This method has no parameters.

Description
Retrieves the Time.

Return
GetTime() returns a CTTIME object with the Time.

See also
SetTime(), CurrentTime()
CTTime::Hour

Syntax
NINT Hour( ) const

Parameters
This method has no parameters.

Description
Retrieves the hour component from a CTTIME object.

Return
Hour returns the unpacked hour.

See also
Minute(), Second(), Unpack(), Pack()
CTTime::Minute

Syntax
NINT Minute( ) const

Parameters
This method has no parameters.

Description
Retrieves the minute component from a CTTIME object

Return
Minute returns the unpacked minute.

See also
Unpack(), Hour(), Second(), Pack()
CTTable::MoveSegment

Moves a key segment to a location.

Declaration

void CTTable::MoveSegment(CTIndex hIndex, NINT segNumber, NINT newIndex);
void CTTable::MoveSegment(NINT indexNumber, NINT segNumber, NINT newIndex);
void CTTAble::MoveSegment(const CTString& indexName, NINT segNumber,
                        NINT newIndex);

Description

CTTable::MoveSegment() moves a key segment to a location indicated by newIndex.

- **hIndex** is an index object.
- **indexNumber** is a relative number for a index object in a table definition.
- **indexName** is the unique name for the index.
- **newIndex** indicates the relative position were the key segment should be moved to.

Return

void

Example

// move the last segment to first
CTIndex hIndex = hTable.GetIndex(0);
NINT count = hIndex.GetSegmentCount();

if (count > 0)
    hTable.MoveSegment(hIndex, (count - 1), 0);

See Also

CTIndex::MoveSegment(), CTSegment::MoveSegment()
### CTTime::Pack

**Syntax**

```cpp
void Pack(NINT hour, NINT minute, NINT second)
```

**Parameters**

- `hour` [in] The hour to pack.
- `minute` [in] The minute to pack.
- `second` [in] The second to pack.

**Description**

Packs an hour, minute and second value into a `CTTIME` object.

**Return**

None.

**See also**

`Unpack()`
**CTTable::Remove**

Removes any c-treeDB table.

**Declaration**

```cpp
void CTTable::Remove();
```

**Description**

The `CTTable::Remove()` function allows any table to be deleted, including tables that are not members of a database.

The `CTTable::Remove()` function deletes a c-tree data file and the associated index files from disk. If the table was opened under a database handle, the table is closed and `CTDatabase::DeleteTable()` is called. If the handle is not active, the table is opened exclusive and then deleted. If the table handle is not active, you must set the path, file extension and password for the handle before calling the `RemoveTable()` method.

A `CTException` is thrown when an error occurs. Use `CTException::GetErrorCode()` and `CTException::GetErrorMsg()` to retrieve the error code and descriptive message.

**Return**

```cpp
void
```

**Example**

```cpp
/* delete a c-treeDB Table */
try {
    MyTable.Remove();
} catch(CTException &E) {
    printf("MyTable.Remove error: (%d) - %s\n", E.GetErrorCode(), E.GetErrorMsg());
}
```

**See Also**

`CTDatabase::DeleteTable()`
CTTime::Second

Syntax
NINT Second() const

Parameters
This method has no parameters.

Description
Retrieves the second component from a CTTIME object.

Return
Second returns the unpacked second.

See also
Minute(), Hour(), SecondOfWeek(), Unpack(), Pack()
CTTable::SetTableKSeg

Establishes a table wide extended key segment definition.

Declaration

void CTTable::SetTableKSeg(pctKSEGDEF pKSeg);

Description

CTTable::SetTableKSeg() establishes a table wide extended key segment definition. pKSeg is a pointer to an extended key segment definition structure with the extended key definition.

Return

void

Example

cTKSEGDEF kseg;

kseg.kseg_ssz = ctKSEG_SSIZ_COMPUTED;
kseg.kseg_type = ctKSEG_TYPE_UNICODE;
kseg.kseg_styp = ctKSEG_STYP_UTF16;
kseg.kseg_comp = ctKSEG_COMPU_S_DEFAULT | ctKSEG_COMPU_N_NONE;
kseg.kseg_desc = "en_US"

hTable.SetTableKSeg(&kseg);

See Also

CTIndex::GetIndexKSeg(), CTIndex::SetIndexKSeg(), CTSegment::GetSegmentKSeg(), CTSegment::SetKSegDefaults(), CTSegment::SetSegmentKSeg(), CTTable::GetTableKSeg()
CTTime::SetTime

Syntax
void SetTime(CTTIME Time)
void SetTime(const CTTime& Time)

Parameters
- *Time* [in] The new Time to set to the *CTTime* object.

Description
Sets the Time to the *CTTime* object.

Return
None.

See also
GetTime()
CTTime::StringToTime

Syntax
void StringToTime(const CTString& str, CTTIME_TYPE TimeType)

Parameters
- `str` [in] The string object to be converted.
- `TimeType` [in] The Time type format to be used in the conversion. Valid Time formats are shown in "Data Types" (page 163).

Description
Converts a `CTString` object to a `Time` object.

Return
None.

See also
TimeToString()
**CTTable::SystemFilterOff**

Deletes a permanent system wide record filter.

**Declaration**

```cpp
void CTTable::SystemFilterOff(CTSYSFILTER mode);
```

**Description**

Deletes a permanent system wide record filter. The table must be opened exclusive and the user must have file definition permission for the table.

*mode* specifies one of the following:

- `CTSYSFILTER_READ` - Indicates you are deleting only a system wide read record filter.
- `CTSYSFILTER_WRITE` - Indicates you are deleting only a system wide write record filter
- `CTSYSFILTER_READ | CT SYSFILTER_WRITE` - Indicates you are deleting both a system wide read record filter and a system wide write record filter.

**Return**

On error `CTTable::SystemFilterOff()` throws a `CTException` exception object.

**See Also**

`CTTable::SystemFilterOn()`
CTTable::SystemFilterOn

Establishes a permanent system wide data record filter.

**Declaration**

```cpp
void CTTable::SystemFilterOn(CTSYSFILTER mode);
```

**Description**

Establishes a permanent system wide, i.e. the filter applies to all users, read and/or write record filter. Depending on server file security setting, the table must be opened exclusive and the user must have file definition permission for the table. Once a read or a write filter is established, it can only be deleted by calling method `CTTable::SystemFilterOff()`.

A table may have at most one read and one write system wide filter. A write filter will be called when data records are added, updated or deleted.

`mode` specifies one of the following:

- `CTSYSFILTER_READ` - Indicates you are deleting only a system wide read record filter.
- `CTSYSFILTER_WRITE` - Indicates you are deleting only a system wide write record filter
- `CTSYSFILTER_READ | CTSYSFILTER_WRITE` - Indicates you are deleting both a system wide read record filter and a system wide write record filter.

System wide filters must be callback filters. The actual callback evaluation takes place in a new callback function `ctfiltercb_rowl()` located in module `ctclbk.c`.

There different levels of security settings when users modify data file definition resources such as `IFIL` and `DODA`. The c-tree Server can be configured for three different levels of data file resource security:

FILEDEF_SECURITY_LEVEL LOW:
Lowest security setting. There is no protection as any user may add or delete data file definition resources. This setting may be used to keep the c-tree Server data compatible with legacy applications.

FILEDEF_SECURITY_LEVEL MEDIUM:
Default security setting. Any user may add or delete data file definition resources, but the file must be opened exclusive. This default setting may be enough to keep the c-tree Server data compatible with most legacy applications.

FILEDEF_SECURITY_LEVEL HIGH:
Highest security setting. A user must have file definition permission before a definition resource is added or deleted. The file must be opened exclusive. This setting is appropriate for applications that require the highest level of security and may cause compatibility problems with existing legacy applications.

**Return**

On error `CTTable::SystemFilterOn()` throws a `CTException` exception object.

**Example**
try
{
    // open an existing table
    hTable.Open("table_name", CTOPEN_EXCLUSIVE);
    // establish a new system wide read-write filter
    hTable.SystemFilterOn(CTSYSFILTER_READ | CTSYSFILTER_WRITE);
    // close the table
    hTable.Close();
}
catch (CTException &err)
{
    printf("Setting system filter failed with error %d\n", err.GetErrorCode());
}
**CTTime::TimeToString**

**Syntax**

```cpp
CTString TimeToString(CTTIME_TYPE TimeType) const
```

**Parameters**

- **TimeType** [in] The Time type format used to convert to string. The valid Time type formats are shown in "Data Types" (page 163).

**Description**

Converts a packed `CTTIME` object to a `CTString` object.

**Return**

`TimeToString()` returns a `CTString` object with the Time.

**See also**

`StringToTime()`
CTTime::Unpack

Syntax
void Unpack(NINT& hour, NINT& minute, NINT& second) const

Parameters
- *hour* [out] The unpacked hour.
- *minute* [out] The unpacked minute.
- *second* [out] The unpacked second.

Description
Unpacks a *CTTIME* object into hour, minute and second values

Return
None.

See also
Pack(), Hour(), Minute(), Second(), DayOfWeek()
### A. c-treeACE Error Code Reference

Please note that several of these "error" codes are not really errors, but rather indicators of an unsuccessful operation. For example, KDUP_ERR (2) occurs if you attempt to add an existing entry to an index that does not support duplicate key values. This is not an error, but a situation to which your application program must be prepared to react.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO_ERROR</td>
<td>Successful operation.</td>
</tr>
<tr>
<td>2</td>
<td>KDUP_ERR</td>
<td>Key value already exists in index.</td>
</tr>
<tr>
<td>3</td>
<td>KMAT_ERR</td>
<td>Could not delete target key value since recbyt does not match associated data record position in index.</td>
</tr>
<tr>
<td>4</td>
<td>KDEL_ERR</td>
<td>Could not find target key value in index. No deletion performed. May indicate improper use of buffers during ReWriteRecord().</td>
</tr>
<tr>
<td>5</td>
<td>KBLD_ERR</td>
<td>Cannot call DeleteKeyBlind() with an index that supports duplicate keys.</td>
</tr>
<tr>
<td>6</td>
<td>BJMP_ERR</td>
<td>ctree() function jump table error.</td>
</tr>
<tr>
<td>7</td>
<td>TUSR_ERR</td>
<td>Terminate user.</td>
</tr>
<tr>
<td>-8</td>
<td>FCNF_COD</td>
<td>This is a sysiocod value when FNOP_ERR was caused by conflicting open requests (Server).</td>
</tr>
<tr>
<td>-9</td>
<td>FDEV_COD</td>
<td>This is a sysiocod value when FNOP_ERR, DCRAT_ERR, or KCRAT_ERR were caused by device access error.</td>
</tr>
<tr>
<td>10</td>
<td>SPAC_ERR</td>
<td>InitCTree() parameters require too much space.</td>
</tr>
<tr>
<td>11</td>
<td>SPRM_ERR</td>
<td>Bad parameter(s): either bufs &lt; 3, idxs &lt; 0, sect &lt;1, or dats &lt; 0.</td>
</tr>
</tbody>
</table>
| 12    | FNOP_ERR          | Could not open file. Either file does not exist, filnam points to incorrect file name, or file is locked by another process. Check sysiocod for the system-level error. For ISAM functions, check isam_fil for the specific file number. 
For the client/server model only, if a file open returns FNOP_ERR, check sysiocod. If sysiocod = FCNF_COD, (-8), the file exists but there is file mode conflict preventing the file from being opened. For example, requesting an cEXCLUSIVE open when the file is already open cISHARED.

The failure to open the file with system error 32 could happen due to third-party backup software having the file open even if it does not lock regions of the file.

For example, if the software has the file open in exclusive mode, an attempt by c-tree to open the file in shared or exclusive mode will fail. If the software has the file open in shared mode, an attempt by c-tree to open the file in exclusive mode will fail. |
<p>| 13    | FUNK_ERR          | OpenIFile() cannot determine type of file. Version 3.3 files must be rebuilt before they can be used. |</p>
<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| 14    | FCRP_ERR         | File appears corrupt at open. This occurs if a file is updated and the disk protocol is set at `NOTFORCE` and `CloseFile()` is not executed. Files have been updated but not properly closed; they were not processed by automatic recovery so they are in an unknown (inconsistent) state. If your transaction logs are corrupted, preventing automatic recovery from occurring, you can either:  
  - Restore from a backup and reapply changes if available (e.g., from application log or forward roll of good transaction logs you have saved).  
  or  
  - Rebuild the files, which will clear the error 14 but will still leave the files in a possibly inconsistent state. In this situation the files will not be guaranteed to be consistent as of any point in time; they can contain a mixture of old/new data, and the data files may not match the index files, due to caching. |
<p>| 15    | FCMP_ERR         | Data file has been compacted, but not rebuilt. Rebuild data file, but do not force rebuild. |
| 16    | KCRAT_ERR        | Could not create index file. Either no space is available on disk or <code>filnam</code> points to improper name. |
| 17    | DCRAT_ERR        | Could not create data file. Either no space is available on disk or <code>filnam</code> points to improper name. |
| 18    | KOPN_ERR         | Tried to create existing index file. |
| 19    | DOPN_ERR         | Tried to create existing data file. |
| 20    | KMIN_ERR         | Key length too large for node size. There must be room for at least 3 key values per node. The node size is given by <code>sect * 128</code> where sect is 3rd parameter. |
| 21    | DREC_ERR         | Cannot create data file with record length smaller than 5 (9 for a huge file). |
| 22    | FNUM_ERR         | <code>filno</code> out of range: <code>0 &lt;= filno &lt; fils</code>, where <code>fils</code> is 2nd parameter. This error may occur if c-tree has not been initialized. |
| 23    | KMEM_ERR         | Illegal index member number. |
| 24    | FCLS_ERR         | Could not close file. Usually indicates that memory is clobbered. |
| 25    | KLNK_ERR         | Bad link in deleted node list. Rebuild file. |
| 26    | FACS_ERR         | File number (<code>datno</code>, <code>keyno</code>, or <code>filno</code>) is not in use. |
| 27    | LBOF_ERR         | Data record position before 1st actual data record |
| 28    | ZDRN_ERR         | <code>AddKey()</code> called with <code>recbyt = 0</code>. |
| 29    | ZREC_ERR         | Data file routine called with <code>recbyt = 0</code>. |
| 30    | LEOF_ERR         | <code>recbyt</code> exceeds logical end of file. If <code>recbyt</code> is correct, then data file header record may be incorrect. If so, rebuild data file. |</p>
<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>DELFLG_ERR</td>
<td>Next record in delete chain of a fixed-length data file does not have 1st byte set to 0xff. Data file header record may be corrupt; if so, rebuild data file. Or C255 constant is incorrect: see ctcmpf.h.</td>
</tr>
<tr>
<td>32</td>
<td>DDRN_ERR</td>
<td>Attempt to delete data record twice in a row.</td>
</tr>
<tr>
<td>33</td>
<td>DNUL_ERR</td>
<td><code>recptr</code> is NULL.</td>
</tr>
<tr>
<td>34</td>
<td>PRDS_ERR</td>
<td>Could not find correct predecessor node. Should not show up in a single user system. Indicates that an index insertion was interrupted before completion. Rebuild index.</td>
</tr>
<tr>
<td>35</td>
<td>SEEK_ERR</td>
<td><code>lseek()</code> failed in function <code>ctio()</code> (<code>ctplib</code> c). Possible causes are: out of disk space, corrupted record position in file, or corrupted file descriptor.</td>
</tr>
<tr>
<td>36</td>
<td>READ_ERR</td>
<td>Read failed in function <code>ctio()</code> (<code>ctplib</code> c). Possible cause: corrupted data record position in file.</td>
</tr>
<tr>
<td>37</td>
<td>WRITE_ERR</td>
<td>Write failed in function <code>ctio()</code>. See 35 above.</td>
</tr>
<tr>
<td>38</td>
<td>VRTO_ERR</td>
<td>Could not convert a virtually opened file to an actually opened file. This might occur if your application uses up some file descriptors after a virtual file has been automatically closed. You can protect against this by lowering the <code>MAXVFIL</code> parameter in <code>ctoptn.h</code>.</td>
</tr>
<tr>
<td>39</td>
<td>FULL_ERR</td>
<td>The 4-byte data record position (or node position) address space has been exhausted.</td>
</tr>
<tr>
<td>40</td>
<td>KSIZ_ERR</td>
<td>The index node size was larger when the index was created. For client/server applications, check your <code>PAGE_SIZE</code> setting in <code>ctsrvr.cfg</code>. For stand-alone applications, check your <code>sect</code> setting (recall that <code>PAGE_SIZE = sect * 128</code>, where <code>sect</code> is the third <code>InitCTree()</code> parameter).</td>
</tr>
<tr>
<td>41</td>
<td>UDLK_ERR</td>
<td>Could not unlock data record. If dummy lock file is in use, be sure it has a file mode of 3.</td>
</tr>
<tr>
<td>42</td>
<td>DLOK_ERR</td>
<td>Could not obtain data record lock.</td>
</tr>
<tr>
<td>43</td>
<td>FVER_ERR</td>
<td>Current configuration parameters are inconsistent with the configuration parameters at the time of file creation.</td>
</tr>
<tr>
<td>44</td>
<td>OSRL_ERR</td>
<td>Data file serial number overflow.</td>
</tr>
<tr>
<td>45</td>
<td>KLEN_ERR</td>
<td>Index key length exceeds <code>MAXLEN</code> parameter. Change <code>MAXLEN</code> in <code>ctoptn.h</code> and recompile c-tree.</td>
</tr>
<tr>
<td>46</td>
<td>FUSE_ERR</td>
<td>File number is already in use. If possible, increase the FILES setting in the server (or the <code>fils</code> parameter of <code>INITISAM</code>) for standalone. In releases after V10.3, the default for <code>MAX_FILES_PER_USER</code> has been increased from 2048 to 32767 to avoid file opens failing due to the lower limit.</td>
</tr>
<tr>
<td>47</td>
<td>FINT_ERR</td>
<td>c-tree has not been initialized</td>
</tr>
<tr>
<td>48</td>
<td>FMOD_ERR</td>
<td>A function has been called for the wrong type of file: e.g., a variable-length function is called for a fixed-length data file.</td>
</tr>
<tr>
<td>49</td>
<td>FSAV_ERR</td>
<td>Could not write file directory updates to disk during file extension.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>50</td>
<td>LNOD_ERR</td>
<td>Could not lock index file node.</td>
</tr>
<tr>
<td>51</td>
<td>UNOD_ERR</td>
<td>Could not unlock index file node. If a dummy lock file is in use, be sure it has a file mode of 3.</td>
</tr>
<tr>
<td>52</td>
<td>KTYP_ERR</td>
<td>Variable-length and/or floating point keys disabled in ctoptn.h.</td>
</tr>
<tr>
<td>53</td>
<td>FTYP_ERR</td>
<td>The file’s file mode is inconsistent with the compile time options selected in ctoptn.h.</td>
</tr>
<tr>
<td>54</td>
<td>REDF_ERR</td>
<td>Attempt to write a read only file.</td>
</tr>
<tr>
<td>55</td>
<td>DLTF_ERR</td>
<td>File deletion failed.</td>
</tr>
<tr>
<td>56</td>
<td>DLTP_ERR</td>
<td>File must be opened exclusive for delete.</td>
</tr>
<tr>
<td>57</td>
<td>DADV_ERR</td>
<td>Proper lock is not held (ctCHECKLOCK/READ).</td>
</tr>
<tr>
<td>58</td>
<td>KLOD_ERR</td>
<td>LoadKey() called with incorrect key number. You cannot continue.</td>
</tr>
<tr>
<td>59</td>
<td>KJOR_ERR</td>
<td>LoadKey() called with key out of order. You may skip this key and continue.</td>
</tr>
<tr>
<td>60</td>
<td>KFRC_ERR</td>
<td>Percent out of range.</td>
</tr>
<tr>
<td>61</td>
<td>CTNL_ERR</td>
<td>NULL file control block detected during I/O.</td>
</tr>
<tr>
<td>62</td>
<td>LERR_ERR</td>
<td>File must be opened exclusively.</td>
</tr>
<tr>
<td>63</td>
<td>RSER_ERR</td>
<td>Start file/log file serial number error.</td>
</tr>
<tr>
<td>64</td>
<td>RLEN_ERR</td>
<td>Checkpoint past end of log file (a checkpoint file is pointing to a region beyond the end of the transaction log). A serious hardware failure can cause the final entries of the transaction log to not be written to disk, causing this error. Try restarting the server with the following keyword in place: CHECKPOINT_PREVIOUS YES</td>
</tr>
<tr>
<td>65</td>
<td>RMEM_ERR</td>
<td>Not enough memory during transaction processing.</td>
</tr>
<tr>
<td>66</td>
<td>RCHK_ERR</td>
<td>Log file entry failed to find checkpoint.</td>
</tr>
<tr>
<td>67</td>
<td>RENF_ERR</td>
<td>Could not rename file.</td>
</tr>
<tr>
<td>68</td>
<td>LALC_ERR</td>
<td>Could not allocate memory for control list.</td>
</tr>
<tr>
<td>69</td>
<td>BNOD_ERR</td>
<td>Node does not belong to index.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>70</td>
<td>TEXS_ERR</td>
<td>Transaction already pending.</td>
</tr>
<tr>
<td>71</td>
<td>TNON_ERR</td>
<td>No active transaction.</td>
</tr>
<tr>
<td>72</td>
<td>TSHD_ERR</td>
<td>No space for shadow buffer.</td>
</tr>
<tr>
<td>73</td>
<td>TLOG_ERR</td>
<td>ctLOGFIL encountered during shadow only.</td>
</tr>
<tr>
<td>74</td>
<td>TRAC_ERR</td>
<td>Recovery: two active transactions for user.</td>
</tr>
<tr>
<td>75</td>
<td>TROW_ERR</td>
<td>Recovery: bad transaction owner.</td>
</tr>
<tr>
<td>76</td>
<td>TBAD_ERR</td>
<td>Recovery: bad transaction type.</td>
</tr>
<tr>
<td>77</td>
<td>TRNM_ERR</td>
<td>Recovery: file name too long.</td>
</tr>
<tr>
<td>78</td>
<td>TABN_ERR</td>
<td>Transaction abandoned: Too many log extents or dynamic dump wait exhausted.</td>
</tr>
<tr>
<td>79</td>
<td>FLOG_ERR</td>
<td>Could not log file opn/cre/cls/del.</td>
</tr>
<tr>
<td>80</td>
<td>BKEY_ERR</td>
<td>NULL target or bad keyno.</td>
</tr>
<tr>
<td>81</td>
<td>ATRN_ERR</td>
<td>Transaction allocation error.</td>
</tr>
<tr>
<td>82</td>
<td>UALC_ERR</td>
<td>User allocation error.</td>
</tr>
<tr>
<td>83</td>
<td>IALC_ERR</td>
<td>ISAM allocation error.</td>
</tr>
<tr>
<td>84</td>
<td>MUSR_ERR</td>
<td>Maximum users exceeded.</td>
</tr>
<tr>
<td>85</td>
<td>LUPD_ERR</td>
<td>Attempt to reduce write lock to read lock after update.</td>
</tr>
<tr>
<td>86</td>
<td>DEAD_ERR</td>
<td>Dead lock detected.</td>
</tr>
<tr>
<td>87</td>
<td>QIET_ERR</td>
<td>System is busy: files in use.</td>
</tr>
<tr>
<td>88</td>
<td>LMEM_ERR</td>
<td>Linked list memory allocation error.</td>
</tr>
<tr>
<td>89</td>
<td>TMEM_ERR</td>
<td>Memory allocation during transaction processing.</td>
</tr>
<tr>
<td>90</td>
<td>NQUE_ERR</td>
<td>Could not create queue.</td>
</tr>
<tr>
<td>91</td>
<td>QWRT_ERR</td>
<td>Queue write error.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>92</td>
<td>QMRT_ERR</td>
<td>Queue memory error during write.</td>
</tr>
<tr>
<td>93</td>
<td>QRED_ERR</td>
<td>Queue read error.</td>
</tr>
<tr>
<td>94</td>
<td>PNDG_ERR</td>
<td>Pending error: Cannot save or commit transaction.</td>
</tr>
<tr>
<td>95</td>
<td>STSK_ERR</td>
<td>Could not start task.</td>
</tr>
<tr>
<td>96</td>
<td>LOPN_ERR</td>
<td>Start-file/log open error.</td>
</tr>
<tr>
<td>97</td>
<td>SUSR_ERR</td>
<td>Bad user handle.</td>
</tr>
<tr>
<td>98</td>
<td>BTMD_ERR</td>
<td>Bad transaction mode.</td>
</tr>
<tr>
<td>99</td>
<td>TTYP_ERR</td>
<td>Transaction type / mode conflict.</td>
</tr>
<tr>
<td>100</td>
<td>ICUR_ERR</td>
<td>No current ISAM record for data file <code>isam_fil</code>.</td>
</tr>
<tr>
<td>101</td>
<td>INOT_ERR</td>
<td>Could not satisfy ISAM search request for index <code>isam_fil</code>. The following items are the most probable causes of the INOT_ERR (101):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Passing <code>GetRecord()</code> a duplicate allowed index number (<code>keyno</code>). <code>GetRecord()</code> does not support duplicate allowed indices.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Not calling <code>TransformKey()</code> on target. Refer to “TransformKey” in the Function Reference Guide.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Passing <code>ctDeleteSequence()</code> a sequence name that does not exist.</td>
</tr>
<tr>
<td>102</td>
<td>INOD_ERR</td>
<td>ISAM parameter file does not exist. Be sure that <code>filnam</code> parameter points to correct name. <code>isam_fil</code> value is undefined.</td>
</tr>
<tr>
<td>103</td>
<td>IGIN_ERR</td>
<td>Could not read ISAM parameter file Initialization record. <code>isam_fil</code> is undefined. Be sure that parameter file is not empty, and that the correct short integer input conversion character has been specified in <code>ctoptn.h</code>.</td>
</tr>
<tr>
<td>104</td>
<td>IFIL_ERR</td>
<td>Number of files opened exceeds <code>fils</code> parameter at initialization. Increase <code>fils</code>. Optionally change <code>ctMAXFIL</code> in <code>ctoptn.h</code>. <code>isam_fil</code> is undefined.</td>
</tr>
<tr>
<td>105</td>
<td>IUND_ERR</td>
<td>Could not undo a rejected ISAM update. Data file must be rebuilt. (During the rebuild, look for records with rejected duplicate key values.)</td>
</tr>
<tr>
<td>106</td>
<td>IDRI_ERR</td>
<td>Could not read ISAM parameter file Data File Description record for <code>isam_fil</code>. Be sure parameter file is consistent with <code>RTREE</code> setting in <code>ctoptn.h</code>.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>107</td>
<td>IDRK_ERR</td>
<td>Too many indices for data file number <code>isam_fil</code> in ISAM parameter file. Change <code>MAX_DAT_KEY</code> in <code>ctoptn.h</code>.</td>
</tr>
<tr>
<td>108</td>
<td>IMKY_ERR</td>
<td><code>keyno</code> for index file member out of sequence. <code>keyno</code> must equal host index file <code>keyno</code> plus member number.</td>
</tr>
<tr>
<td>109</td>
<td>IKRS_ERR</td>
<td>Too many key segments defined in ISAM parameter file for index number <code>isam_fil</code>. Change <code>MAX_KEY_SEG</code> in <code>ctoptn.h</code>. Be sure that parameter file is consistent with <code>RTREE</code> setting in <code>ctoptn.h</code>.</td>
</tr>
<tr>
<td>110</td>
<td>ISRC_ERR</td>
<td>Could not read ISAM parameter file Key Segment Description record for index number <code>isam_fil</code>.</td>
</tr>
<tr>
<td>111</td>
<td>IKRI_ERR</td>
<td>Could not read ISAM parameter file Index File Description record. <code>isam_fil</code> indicates the relative index number for an unspecified data file.</td>
</tr>
<tr>
<td>112</td>
<td>IPND_ERR</td>
<td><code>(ctENABLE)</code> found pending locks.</td>
</tr>
<tr>
<td>113</td>
<td>INOL_ERR</td>
<td>No space left in c-tree’s internal lock list.</td>
</tr>
<tr>
<td>114</td>
<td>IRED_ERR</td>
<td>1st byte of fixed-length data record found by ISAM routine equals delete flag.</td>
</tr>
<tr>
<td>115</td>
<td>ISLN_ERR</td>
<td>Sum of key segment lengths does not match key length for index number <code>isam_fil</code>.</td>
</tr>
<tr>
<td>116</td>
<td>IMOD_ERR</td>
<td>Bad mode parameter.</td>
</tr>
<tr>
<td>117</td>
<td>IMRI_ERR</td>
<td>Could not read ISAM parameter file Index Member record.</td>
</tr>
<tr>
<td>118</td>
<td>SKEY_ERR</td>
<td><code>NextInSet</code> or <code>PreviousInSet</code> called with a keyno that does not match keyno in last call to <code>FirstInSet</code> or .</td>
</tr>
<tr>
<td>119</td>
<td>SKTY_ERR</td>
<td><code>FirstInSet</code> called for numeric key type.</td>
</tr>
<tr>
<td>120</td>
<td>RRLN_ERR</td>
<td>Not enough dynamic memory for record buffer in <code>ctrbld.c</code>.</td>
</tr>
<tr>
<td>121</td>
<td>KBUF_ERR</td>
<td>Tried to update data with <code>ctISAMKBUFhdr</code> on.</td>
</tr>
<tr>
<td>122</td>
<td>RMOD_ERR</td>
<td>Attempt to change between fixed and variable-length records during rebuild. Once a data file is created, its record length characteristic cannot be changed.</td>
</tr>
<tr>
<td>123</td>
<td>RVHD_ERR</td>
<td>A variable-length data record is not preceded by a valid record mark. The file is apparently corrupted.</td>
</tr>
<tr>
<td>124</td>
<td>INIX_ERR</td>
<td>Number of indices in index file does not match <code>IFIL</code> structure in call to <code>OpenFile()</code> or the parameter file specified by .</td>
</tr>
<tr>
<td>125</td>
<td>IINT_ERR</td>
<td>c-tree is already initialized via a previous call.</td>
</tr>
<tr>
<td>126</td>
<td>ABDR_ERR</td>
<td>Bad directory path get.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>128</td>
<td>ARSP_ERR</td>
<td>Could not receive answer. See &quot;Client/Server ctnrio Communication Errors (Formerly VDP Errors)&quot; for more information.</td>
</tr>
<tr>
<td>129</td>
<td>NINT_ERR</td>
<td>c-tree not initialized.</td>
</tr>
<tr>
<td>130</td>
<td>AFNM_ERR</td>
<td>NULL file name pointer. Empty file name on call to open or create.</td>
</tr>
<tr>
<td>131</td>
<td>AFLN_ERR</td>
<td>File name length exceeds message size.</td>
</tr>
<tr>
<td>132</td>
<td>ASPC_ERR</td>
<td>No room for application message buffer.</td>
</tr>
<tr>
<td>133</td>
<td>ASKY_ERR</td>
<td>Could not identify Server.</td>
</tr>
<tr>
<td>134</td>
<td>ASID_ERR</td>
<td>Could not get Servers message ID.</td>
</tr>
<tr>
<td>135</td>
<td>AAID_ERR</td>
<td>Could not allocate application ID.</td>
</tr>
<tr>
<td>136</td>
<td>AMST_ERR</td>
<td>Could not get application message status.</td>
</tr>
<tr>
<td>137</td>
<td>AMQZ_ERR</td>
<td>Could not set application message size.</td>
</tr>
<tr>
<td>138</td>
<td>AMRD_ERR</td>
<td>Could not get rid of application message.</td>
</tr>
<tr>
<td>139</td>
<td>ABNM_ERR</td>
<td>Badly formed file name.</td>
</tr>
<tr>
<td>140</td>
<td>VMAX_ERR</td>
<td>Variable record length exceeds 65,529 bytes.</td>
</tr>
<tr>
<td>141</td>
<td>AMSG_ERR</td>
<td>Required message size exceeds maximum.</td>
</tr>
<tr>
<td>142</td>
<td>SMXL_ERR</td>
<td>Application MAXLEN &gt; Server's MAXLEN ctoptn.h.</td>
</tr>
<tr>
<td>143</td>
<td>SHND_ERR</td>
<td>Communications handler not installed.</td>
</tr>
<tr>
<td>144</td>
<td>QMEM_ERR</td>
<td>Application could not ID output queue.</td>
</tr>
<tr>
<td>145</td>
<td>ALOG_ERR</td>
<td>No message space. Was login ok?</td>
</tr>
<tr>
<td>146</td>
<td>VDLK_ERR</td>
<td>Could not update available space information in variable-length data file.</td>
</tr>
<tr>
<td>147</td>
<td>VDLFLG_ERR</td>
<td>Record pointed to by available space information is not marked deleted in variable-length data file.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>148</td>
<td>VLEN_ERR</td>
<td>Attempt to write a variable-length record into a file position which is too small for the record.</td>
</tr>
<tr>
<td>149</td>
<td>VRLN_ERR</td>
<td>Variable-length passed to <code>AddVRecord()</code> is less than minimum record length established at file creation.</td>
</tr>
<tr>
<td>150</td>
<td>SHUT_ERR</td>
<td>Server is shutting down.</td>
</tr>
<tr>
<td>151</td>
<td>STRN_ERR</td>
<td>Could not shut down; transactions pending.</td>
</tr>
<tr>
<td>152</td>
<td>LEXT_ERR</td>
<td>Could not extend logfile.</td>
</tr>
<tr>
<td>153</td>
<td>VBSZ_ERR</td>
<td>Buffer in call to <code>ReReadVRecord()</code> is too small for the variable-length record.</td>
</tr>
<tr>
<td>154</td>
<td>VRCL_ERR</td>
<td>Attempt to read (R) a zero length record from a variable-length data file.</td>
</tr>
<tr>
<td>155</td>
<td>SYST_ERR</td>
<td>Native system failure.</td>
</tr>
<tr>
<td>156</td>
<td>NTIM_ERR</td>
<td>Timeout error.</td>
</tr>
<tr>
<td>158</td>
<td>VFLG_ERR</td>
<td><code>ReReadVRecord()</code> record not marked active.</td>
</tr>
<tr>
<td>159</td>
<td>VPNT_ERR</td>
<td>Zero data record position in variable-length function.</td>
</tr>
<tr>
<td>160</td>
<td>ITIM_ERR</td>
<td>Multi-user interference: key value changed between index search and subsequent data record read.</td>
</tr>
<tr>
<td>161</td>
<td>SINA_ERR</td>
<td>User appears inactive.</td>
</tr>
<tr>
<td>162</td>
<td>SGON_ERR</td>
<td>Server has gone away.</td>
</tr>
<tr>
<td>163</td>
<td>SFRE_ERR</td>
<td>No more room in Server lock table - free up memory.</td>
</tr>
<tr>
<td>164</td>
<td>SFIL_ERR</td>
<td>File number out of range.</td>
</tr>
<tr>
<td>165</td>
<td>SNFB_ERR</td>
<td>No c-tree file control block. Try increasing the FILES limit in <code>ctsvr.cfg</code> and restarting c-treeACE Server. (In standalone mode, limit is set by <code>InitISAM fils</code> parameter.)</td>
</tr>
<tr>
<td>166</td>
<td>SNMC_ERR</td>
<td>No more c-tree file control blocks in Server. Try increasing the FILES limit in <code>ctsvr.cfg</code> and restarting c-treeACE Server. (In standalone mode, limit is set by <code>InitISAM fils</code> parameter.)</td>
</tr>
<tr>
<td>167</td>
<td>SRQS_ERR</td>
<td>Could not read request.</td>
</tr>
<tr>
<td>168</td>
<td>SRSP_ERR</td>
<td>Could not send answer.</td>
</tr>
<tr>
<td>169</td>
<td>TCRE_ERR</td>
<td>Create file already opened (in recovery).</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>170</td>
<td>SFUN_ERR</td>
<td>Bad function number at Server.</td>
</tr>
<tr>
<td>171</td>
<td>SMSG_ERR</td>
<td>Application message size exceeds Server size.</td>
</tr>
<tr>
<td>172</td>
<td>SSPC_ERR</td>
<td>Could not allocate Server message buffer.</td>
</tr>
<tr>
<td>173</td>
<td>SSKY_ERR</td>
<td>Could not identify Server.</td>
</tr>
<tr>
<td>174</td>
<td>SSID_ERR</td>
<td>Could not get Server message ID.</td>
</tr>
<tr>
<td>175</td>
<td>SAMS_ERR</td>
<td>Server could not allocate user message area</td>
</tr>
<tr>
<td>176</td>
<td>SMST_ERR</td>
<td>Could not get Server message status</td>
</tr>
<tr>
<td>177</td>
<td>SMQZ_ERR</td>
<td>Could not set message Server message size</td>
</tr>
<tr>
<td>178</td>
<td>SINM_ERR</td>
<td>Unexpected file number assigned to [si] in recovery.</td>
</tr>
<tr>
<td>179</td>
<td>SOUT_ERR</td>
<td>Server is at full user capacity</td>
</tr>
<tr>
<td>180</td>
<td>IKRU_ERR</td>
<td>Could not read r-tree symbolic index name.</td>
</tr>
<tr>
<td>181</td>
<td>IKMU_ERR</td>
<td>Could not get dynamic memory for r-tree symbolic index name.</td>
</tr>
<tr>
<td>182</td>
<td>IKSR_ERR</td>
<td>Cannot accommodate temporary r-tree files. Increase ctMAXFIL.</td>
</tr>
<tr>
<td>183</td>
<td>IDRU_ERR</td>
<td>Could not read r-tree data field symbolic names.</td>
</tr>
<tr>
<td>184</td>
<td>ISDP_ERR</td>
<td>Multiple set buffer space already allocated.</td>
</tr>
<tr>
<td>185</td>
<td>ISAL_ERR</td>
<td>Not enough dynamic memory for multiple sets.</td>
</tr>
<tr>
<td>186</td>
<td>ISNM_ERR</td>
<td>Set number out of range in .</td>
</tr>
<tr>
<td>187</td>
<td>IRBF_ERR</td>
<td>Null buffer pointer in r-tree.</td>
</tr>
<tr>
<td>188</td>
<td>ITBF_ERR</td>
<td>Null target buffer pointer in r-tree.</td>
</tr>
<tr>
<td>189</td>
<td>IJSK_ERR</td>
<td>JOINS_TO skip condition in r-tree.</td>
</tr>
<tr>
<td>190</td>
<td>IJER_ERR</td>
<td>JOINS_TO error condition in r-tree.</td>
</tr>
<tr>
<td>191</td>
<td>IJNL_ERR</td>
<td>JOINS_TO null fill condition in r-tree.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>192</td>
<td>IDSK_ERR</td>
<td>IS_DETAIL_FOR skip condition in r-tree.</td>
</tr>
<tr>
<td>193</td>
<td>IDER_ERR</td>
<td>IS_DETAIL_FOR error condition in r-tree.</td>
</tr>
<tr>
<td>194</td>
<td>IDNL_ERR</td>
<td>IS_DETAIL_FOR null fill condition in r-tree.</td>
</tr>
<tr>
<td>195</td>
<td>IDMU_ERR</td>
<td>Could not get dynamic memory for r-tree data field symbolic names.</td>
</tr>
<tr>
<td>196</td>
<td>ITML_ERR</td>
<td>Exceeded RETRY_LIMIT on error 160 in r-tree.</td>
</tr>
<tr>
<td>197</td>
<td>IMEM_ERR</td>
<td>Could not get memory for IFIL block.</td>
</tr>
<tr>
<td>198</td>
<td>BIFL_ERR</td>
<td>Improper IFIL block.</td>
</tr>
<tr>
<td>199</td>
<td>NSCH_ERR</td>
<td>Key segment refers to schema but no schema is defined.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>RCRE_ERR</td>
<td>Resource already enabled.</td>
</tr>
<tr>
<td>401</td>
<td>RNON_ERR</td>
<td>Resources not enabled</td>
</tr>
<tr>
<td>402</td>
<td>RXCL_ERR</td>
<td>File must be exclusive to enable resource.</td>
</tr>
<tr>
<td>403</td>
<td>RZRO_ERR</td>
<td>Empty resource ID.</td>
</tr>
<tr>
<td>404</td>
<td>RBUF_ERR</td>
<td>Output buffer to small.</td>
</tr>
<tr>
<td>405</td>
<td>RDUP_ERR</td>
<td>Resource ID already added.</td>
</tr>
<tr>
<td>406</td>
<td>RCSE_ERR</td>
<td>Bad resource search mode.</td>
</tr>
<tr>
<td>407</td>
<td>RRED_ERR</td>
<td>Attempt to get non-resource info. Can occur on fixed-length file with numeric value in first position. See &quot;IMPORTANT - Data Record Delete Flag&quot; for more information.</td>
</tr>
<tr>
<td>408</td>
<td>RNOT_ERR</td>
<td>Resource not found.</td>
</tr>
<tr>
<td>409</td>
<td>RELN_ERR</td>
<td>Resource length &lt;= 0</td>
</tr>
<tr>
<td>410</td>
<td>USTP_ERR</td>
<td>User not active.</td>
</tr>
<tr>
<td>411</td>
<td>BSUP_ERR</td>
<td>Not a superfile.</td>
</tr>
<tr>
<td>412</td>
<td>SOPN_ERR</td>
<td>Attempt to create open superfile member.</td>
</tr>
<tr>
<td>413</td>
<td>SDIR_ERR</td>
<td>Superfile host not opened.</td>
</tr>
<tr>
<td>414</td>
<td>SNST_ERR</td>
<td>Cannot nest superfiles.</td>
</tr>
<tr>
<td>415</td>
<td>SADD_ERR</td>
<td>Illegal AddKey() to superfile.</td>
</tr>
<tr>
<td>416</td>
<td>SDEL_ERR</td>
<td>Illegal DELBLD to superfile.</td>
</tr>
<tr>
<td>417</td>
<td>SPAG_ERR</td>
<td>Superfile created with different index node size.</td>
</tr>
<tr>
<td>418</td>
<td>SNAM_ERR</td>
<td>Superfile created with maximum name length exceeding</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>419</td>
<td>SRCV_ERR</td>
<td>Host superfile does not support recovery.</td>
</tr>
<tr>
<td>420</td>
<td>TPND_ERR</td>
<td>Key update with pending transaction.</td>
</tr>
<tr>
<td>421</td>
<td>BTFL_ERR</td>
<td>Filter not supported. Probably an invalid value in mode parameter.</td>
</tr>
<tr>
<td>422</td>
<td>BTFN_ERR</td>
<td>Function not supported. Probably an invalid value in mode parameter.</td>
</tr>
<tr>
<td>423</td>
<td>BTIC_ERR</td>
<td>Incomplete batch. You specified BAT_COMPLETE in a DoBatch() call.</td>
</tr>
<tr>
<td>424</td>
<td>BTAD_ERR</td>
<td>Add list error. Internal processing problem. May be a memory overwrite.</td>
</tr>
<tr>
<td>425</td>
<td>BTIP_ERR</td>
<td>Batch already in progress. You made a call to DoBatch() to open a new batch before you have completed the prior one. Make a call with a mode of BAT_CAN first.</td>
</tr>
<tr>
<td>426</td>
<td>BTNO_ERR</td>
<td>No batch active. You are calling DoBatch() with a mode of BAT_CAN or BAT_NXT.</td>
</tr>
<tr>
<td>427</td>
<td>BTST_ERR</td>
<td>Status info already returned. You are making two consecutive DoBatch() calls looking just for status information.</td>
</tr>
<tr>
<td>428</td>
<td>BTMT_ERR</td>
<td>No more info; batch cancelled.</td>
</tr>
<tr>
<td>429</td>
<td>BTBZ_ERR</td>
<td>bufsize is too small for a single record. You need a larger buffer for this file.</td>
</tr>
<tr>
<td>430</td>
<td>BTRQ_ERR</td>
<td>Request is a NULL pointer.</td>
</tr>
<tr>
<td>431</td>
<td>LAGR_ERR</td>
<td>Aggregate/serialization lock denied.</td>
</tr>
<tr>
<td>432</td>
<td>FLEN_ERR</td>
<td>Fixed-length string requires len in DODA.</td>
</tr>
<tr>
<td>433</td>
<td>SSCH_ERR</td>
<td>Segment definition inconsistent with schema.</td>
</tr>
<tr>
<td>434</td>
<td>DLEN_ERR</td>
<td>Very long def block not supported.</td>
</tr>
<tr>
<td>435</td>
<td>FMEM_ERR</td>
<td>File def memory error.</td>
</tr>
<tr>
<td>436</td>
<td>DNUM_ERR</td>
<td>Bad def number.</td>
</tr>
<tr>
<td>437</td>
<td>DADR_ERR</td>
<td>defptr NULL during GETDEFBLK().</td>
</tr>
<tr>
<td>438</td>
<td>DZRO_ERR</td>
<td>Resource found with zero (0) length.</td>
</tr>
<tr>
<td>439</td>
<td>DCNV_ERR</td>
<td>File definition conversion missing; call FairCom.</td>
</tr>
<tr>
<td>440</td>
<td>DDDM_ERR</td>
<td>Dynamic dump already in progress.</td>
</tr>
<tr>
<td>441</td>
<td>DMEM_ERR</td>
<td>No memory for dynamic dump file buffer.</td>
</tr>
<tr>
<td>442</td>
<td>DAVL_ERR</td>
<td>One or more files not available for dump.</td>
</tr>
<tr>
<td>443</td>
<td>DSIZ_ERR</td>
<td>File length discrepancy.</td>
</tr>
<tr>
<td>444</td>
<td>DCRE_ERR</td>
<td>Could not create file during dump recovery.</td>
</tr>
<tr>
<td>445</td>
<td>SDAT_ERR</td>
<td>Not enough data to assemble key.</td>
</tr>
<tr>
<td>446</td>
<td>BMOD_ERR</td>
<td>Invalid key mode.</td>
</tr>
<tr>
<td>447</td>
<td>BOWN_ERR</td>
<td>You must be the owner of this file.</td>
</tr>
<tr>
<td>448</td>
<td>DEFP_ERR</td>
<td>File definition permission denied.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>449</td>
<td>DADM_ERR</td>
<td>ADMIN has opened file. Cannot delete file.</td>
</tr>
<tr>
<td>450</td>
<td>LUID_ERR</td>
<td>Invalid user ID when logging on Server.</td>
</tr>
<tr>
<td>451</td>
<td>LPWD_ERR</td>
<td>Invalid password when logging on Server.</td>
</tr>
<tr>
<td>452</td>
<td>LSRV_ERR</td>
<td>Server could not process user/acct info.</td>
</tr>
<tr>
<td>453</td>
<td>NSRV_ERR</td>
<td>Invalid Server name.</td>
</tr>
<tr>
<td>454</td>
<td>NSUP_ERR</td>
<td>Service not supported. Using an option unavailable to this library.</td>
</tr>
<tr>
<td>455</td>
<td>SGRP_ERR</td>
<td>User does not belong to group.</td>
</tr>
<tr>
<td>456</td>
<td>SACS_ERR</td>
<td>Group access denied.</td>
</tr>
<tr>
<td>457</td>
<td>SPWD_ERR</td>
<td>File password invalid.</td>
</tr>
<tr>
<td>458</td>
<td>SWRT_ERR</td>
<td>Write permission not granted.</td>
</tr>
<tr>
<td>459</td>
<td>SDLT_ERR</td>
<td>Delete permission denied</td>
</tr>
<tr>
<td>460</td>
<td>SRES_ERR</td>
<td>Resource not enabled.</td>
</tr>
<tr>
<td>461</td>
<td>SPER_ERR</td>
<td>Bad permission flag.</td>
</tr>
<tr>
<td>462</td>
<td>SHDR_ERR</td>
<td>No superfile directory found during CTSBLD() processing.</td>
</tr>
<tr>
<td>463</td>
<td>UQID_ERR</td>
<td>File does not have unique Server/File number.</td>
</tr>
<tr>
<td>464</td>
<td>IISM_ERR</td>
<td>ISAM level logon not performed.</td>
</tr>
<tr>
<td>465</td>
<td>IIIN_ERR</td>
<td>Incremental Index: $dnumidx &lt; 1$.</td>
</tr>
<tr>
<td>466</td>
<td>IIDT_ERR</td>
<td>Incremental Index: $dfilno$ not an ISAM file.</td>
</tr>
<tr>
<td>467</td>
<td>IIINM_ERR</td>
<td>Incremental Index: $aidxnam$ NULL for 1st.</td>
</tr>
<tr>
<td>468</td>
<td>IITR_ERR</td>
<td>Incremental Index: active tran not allowed.</td>
</tr>
<tr>
<td>469</td>
<td>NGIO_ERR</td>
<td>Negative I/O request.</td>
</tr>
<tr>
<td>470</td>
<td>LGST_ERR</td>
<td>Guest logons disabled.</td>
</tr>
<tr>
<td>471</td>
<td>JOBT_ERR</td>
<td>Job Thread Launch error - ctFeatJOBMGTSYS</td>
</tr>
<tr>
<td>471-497</td>
<td>Sort Errors</td>
<td>See 471-497 Sort Error Codes.</td>
</tr>
<tr>
<td>498</td>
<td>NLOG_ERR</td>
<td>Old log file found during log create.</td>
</tr>
<tr>
<td>499</td>
<td>FIDD_ERR</td>
<td>Mismatch between recovery log and file ID.</td>
</tr>
<tr>
<td>500</td>
<td>SQLINIT_ERR</td>
<td>Server could not init SQL engine.</td>
</tr>
<tr>
<td>501</td>
<td>SQLCONNECT_ERR</td>
<td>Could not init SQL for a user.</td>
</tr>
<tr>
<td>502</td>
<td>SQL_REQUEST_ERROR</td>
<td>Could not access SQL master info.</td>
</tr>
<tr>
<td>503</td>
<td>SQL_INVALID_CONTIN</td>
<td>Could not continue SQL request</td>
</tr>
<tr>
<td>504</td>
<td>NSQL_ERR</td>
<td>Server does not support SQL.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>505</td>
<td>USQL_ERR</td>
<td>User profile does not enable SQL.</td>
</tr>
<tr>
<td>506</td>
<td>SRFL_ERR</td>
<td>Could not open save/restore file.</td>
</tr>
<tr>
<td>507</td>
<td>SRIO_ERR</td>
<td>Could not process save/restore file.</td>
</tr>
<tr>
<td>508</td>
<td>SRIN_ERR</td>
<td>Save/restore inconsistency.</td>
</tr>
<tr>
<td>509</td>
<td>DSRV_ERR</td>
<td>Duplicate Server.</td>
</tr>
<tr>
<td>510</td>
<td>RFCK_ERR</td>
<td>Active chkpnt at start of roll-forward.</td>
</tr>
<tr>
<td>511</td>
<td>ILOP_ERR</td>
<td>Index nodes form illegal loop: rebuild.</td>
</tr>
<tr>
<td>512</td>
<td>DLOP_ERR</td>
<td>Data file loop detected.</td>
</tr>
<tr>
<td>513</td>
<td>SBLF_ERR</td>
<td><em>FPUTFGET</em> does not support <em>CTSBLDX</em> ().</td>
</tr>
<tr>
<td>514</td>
<td>CQUE_ERR</td>
<td>Queue has been closed.</td>
</tr>
<tr>
<td>515</td>
<td>OIFL_ERR</td>
<td>Cannot convert old <em>IFIL</em> structure.</td>
</tr>
<tr>
<td>516</td>
<td>GNUL_ERR</td>
<td><em>ctNOGLOBALS</em> not allocated.</td>
</tr>
<tr>
<td>517</td>
<td>GNOT_ERR</td>
<td><em>regid</em> is not registered.</td>
</tr>
<tr>
<td>518</td>
<td>GEXS_ERR</td>
<td><em>regid</em> is already registered.</td>
</tr>
</tbody>
</table>
| 519   | IEOF_ERR          | Index logical EOF error. The index end of file error gives warning when a B-tree index node appears to be past the logical end of the index file. Such an error indicates that the index is corrupted. Note the following points:  
• Should not occur on indices under transaction processing control or in Standalone Multi-user mode.  
• On non-transaction processed indices, something has corrupted the index. The index must be rebuilt or recovered as follows.  
Recover by copying the key values to a new index. The corrupted index file can be opened with the file modes: *ctOPENCRPT*, *ctREADFIL*, permitting the index to be searched, and the *IEOF_ERR* suppressed. |
<p>| 520   | HTRN_ERR          | Attempt to update index with inconsistent transaction number. If encountered, compact and rebuild the file to resolve the error. |
| 521   | BMAL_ERR          | Could not allocate memory for the Streettalk login message buffer. |
| 522   | STID_ERR          | Userid in <em>InitISAM()</em> does not match current login ID. |
| 527   | BIDX_ERR          | Index must be rebuilt. See <em>CTSTATUS.FCS</em>. |
| 528   | SLEN_ERR          | Key segment length error. |
| 529   | CHKP_ERR          | System checkpoints terminated. |
| 530   | LMTC_ERR          | Client does not match server. |
| 531   | BREP_ERR          | Index reorg entry error. |
| 532   | ASAV_ERR          | <em>SetSavePoint()</em> called with <em>ctAUTOSAVE</em> on |
| 533   | MTRN_ERR          | File header high-water-mark overflow. |
| 534   | OTRN_ERR          | Transaction Number Overflow or Pending File ID Overflow (see <em>Pending File ID Overflow</em> (<a href="http://www.faircom.com/wp/file_id/">http://www.faircom.com/wp/file_id/</a>) in |</p>
<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>535</td>
<td>REGC_ERR</td>
<td>c-tree instance not registered.</td>
</tr>
<tr>
<td>536</td>
<td>AREG_ERR</td>
<td>Only automatic RegisterCTree() allowed.</td>
</tr>
<tr>
<td>537</td>
<td>TCOL_ERR</td>
<td>Transaction log collision.</td>
</tr>
<tr>
<td>538</td>
<td>PIOT_ERR</td>
<td>Client-side bad function array type.</td>
</tr>
<tr>
<td>-539</td>
<td>BFIL_COD</td>
<td>sysiocod when file does not appear to contain any valid information.</td>
</tr>
<tr>
<td>540</td>
<td>PNUL_ERR</td>
<td>NULL parameter</td>
</tr>
<tr>
<td>541</td>
<td>LWRT_ERR</td>
<td>Transaction log cannot be written.</td>
</tr>
<tr>
<td>542</td>
<td>MCRE_ERR</td>
<td>Could not create mirror file.</td>
</tr>
<tr>
<td>543</td>
<td>MOPN_ERR</td>
<td>Could not open mirror file.</td>
</tr>
<tr>
<td>544</td>
<td>MCLS_ERR</td>
<td>Could not close mirror file.</td>
</tr>
<tr>
<td>545</td>
<td>MDLT_ERR</td>
<td>Could not delete mirror file.</td>
</tr>
<tr>
<td>546</td>
<td>MWRT_ERR</td>
<td>Could not write mirror file.</td>
</tr>
<tr>
<td>547</td>
<td>MSAV_ERR</td>
<td>Could not save mirror file.</td>
</tr>
<tr>
<td>548</td>
<td>MRED_ERR</td>
<td>Could not read mirror file.</td>
</tr>
<tr>
<td>549</td>
<td>MHDR_ERR</td>
<td>Mismatch between mirror headers.</td>
</tr>
<tr>
<td>550</td>
<td>MSKP_ERR</td>
<td>Attempt to open primary w/o mirror: OR-ing in a file mode of ctMIRROR_SKP permits a primary file to be opened w/o error.</td>
</tr>
<tr>
<td>551</td>
<td>MNOT_ERR</td>
<td>File already opened without mirror.</td>
</tr>
<tr>
<td>552</td>
<td>MSEG_ERR</td>
<td>Segmented file cannot be mirrored.</td>
</tr>
</tbody>
</table>
| 553   | FBEG_ERR          | Attempt to add a record beginning with either a delete flag (0xFF) or a resource mark (0xFEFE) failed. This capability can be disabled:  
  - For non-server applications, disable the check by compiling with NO_ctBEHAV_CHECKFIX defined.  
  - For client/server applications, the configuration keyword COMPATIBILITY NO_CHECKFIX turns off this check. |
| 554   | ISRL_ERR          | Inconsistent SerialNum definition info.  
  It is inappropriate (regardless of partitioned files or not) to use PRMIIDX to add indices that use SRLSEG if SRLSEG based indices do not already exist. First, a SRLSEG index requires space in the data record. Second, none of the existing records would have SRLSEG data already in place, even if space had been reserved. ISRL_ERR (554) is returned when PRMIIDX attempts to add indices with SRLSEG when SRLSEG has not already been used. |
<p>| 555   | PREA_ERR          | Could not read primary files, so it is switching to mirrored files. |
| 556   | PWRT_ERR          | Could not write primary files, so it is switching to mirrored files. |
| 557   | CWRT_ERR          | Could not write mirror files, so it is suspending mirrored files. |
| 558   | PSAV_ERR          | Could not save primary files, so it is switching to mirrored files. |
| 559   | CSAV_ERR          | Could not save mirror files, so it is suspending mirrored files. |</p>
<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>560</td>
<td>SMON_ERR</td>
<td>Only one of each monitor at a time.</td>
</tr>
<tr>
<td>561</td>
<td>DDMP_BEG</td>
<td>SYSMON: dynamic dump begins.</td>
</tr>
<tr>
<td>562</td>
<td>DDMP_END</td>
<td>SYSMON: dynamic dump ends.</td>
</tr>
<tr>
<td>563</td>
<td>DDMP_ERR</td>
<td>SYSMON: dynamic dump ends (errors).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>At the end of automatic recovery, the following conditions (errors 570-576) may be detected which require cleanup before continuing. The specifics are reported in CTSTATUS.FCS:</td>
</tr>
<tr>
<td>570</td>
<td>RCL1_ERR</td>
<td>• Incomplete compression.</td>
</tr>
<tr>
<td>571</td>
<td>RCL2_ERR</td>
<td>• Index rebuild required.</td>
</tr>
<tr>
<td>572</td>
<td>RCL3_ERR</td>
<td>• Incomplete compression and index rebuild required.</td>
</tr>
<tr>
<td>573</td>
<td>RCL4_ERR</td>
<td>• Primary/mirror files out-of-synchronization. Copy good file over bad.</td>
</tr>
<tr>
<td>574</td>
<td>RCL5_ERR</td>
<td>• Incomplete compression and primary/mirror files out-of-synchronization.</td>
</tr>
<tr>
<td>575</td>
<td>RCL6_ERR</td>
<td>• Index rebuild required and primary/mirror files out-of-synchronization.</td>
</tr>
<tr>
<td>576</td>
<td>RCL7_ERR</td>
<td>• Incomplete compression and index rebuild required and primary/mirror files out-of-synchronization.</td>
</tr>
<tr>
<td>577</td>
<td>DWRT_ERR</td>
<td>Update refused during protected dmp.</td>
</tr>
<tr>
<td>579</td>
<td>LIVL_ERR</td>
<td>Logon interval error.</td>
</tr>
<tr>
<td>580</td>
<td>FSEC_ERR</td>
<td>Could not perform file security op.</td>
</tr>
<tr>
<td>581</td>
<td>EGRP_ERR</td>
<td>Group does not exist.</td>
</tr>
<tr>
<td>582</td>
<td>KSML_ERR</td>
<td>Key length too small.</td>
</tr>
<tr>
<td>583</td>
<td>SAVL_ERR</td>
<td>File extension exceeds diskfull limit.</td>
</tr>
<tr>
<td>584</td>
<td>LRSN_ERR</td>
<td>Exceeded failed logon limit.</td>
</tr>
<tr>
<td>585</td>
<td>LVAL_ERR</td>
<td>Logon date exception.</td>
</tr>
<tr>
<td>-586</td>
<td>MFID_COD</td>
<td>File name different, however, matching file ID found.</td>
</tr>
<tr>
<td>-587</td>
<td>CPND_COD</td>
<td>Close/delete deferred: pending transaction.</td>
</tr>
<tr>
<td>588</td>
<td>CPND_ERR</td>
<td>Attempt to close or delete file with pending transaction.</td>
</tr>
<tr>
<td>589</td>
<td>LADM_ERR</td>
<td>Member of ADMIN group required.</td>
</tr>
<tr>
<td>590</td>
<td>NCON_ERR</td>
<td>Could not find ISAM context ID.</td>
</tr>
<tr>
<td>591</td>
<td>OCON_ERR</td>
<td>Old context ID. Call .</td>
</tr>
<tr>
<td>592</td>
<td>ECON_ERR</td>
<td>Context ID exists.</td>
</tr>
<tr>
<td>593</td>
<td>XUSR_ERR</td>
<td>Non-ADMIN user blocked from log on.</td>
</tr>
<tr>
<td>-594</td>
<td>XUSR_COD</td>
<td>Users in SEC_BLOCK class logged on.</td>
</tr>
<tr>
<td>595</td>
<td>CLEN_ERR</td>
<td>value too small in PUTCREM.</td>
</tr>
<tr>
<td>596</td>
<td>CMIS_ERR</td>
<td>Missing information.</td>
</tr>
<tr>
<td>597</td>
<td>CINI_ERR</td>
<td>Could not initialize expression.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>598</td>
<td>CVAL_ERR</td>
<td>Could not evaluate conditional expression.</td>
</tr>
<tr>
<td>599</td>
<td>DEXT_ERR</td>
<td>Dynamic Dump extent error.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>600</td>
<td>CTHD_ERR</td>
<td>No more client threads.</td>
</tr>
<tr>
<td>601</td>
<td>VRFY_ERR</td>
<td>ctVERIFY detected problems with index.</td>
</tr>
<tr>
<td>602</td>
<td>CMEM_ERR</td>
<td>No memory for system lock table.</td>
</tr>
<tr>
<td>603</td>
<td>FLEX_ERR</td>
<td>Could not allocate FCB.</td>
</tr>
<tr>
<td>604</td>
<td>FINC_ERR</td>
<td>Could not increase user files.</td>
</tr>
<tr>
<td>605</td>
<td>KSRL_ERR</td>
<td>Records with bad (all FF) serial numbers.</td>
</tr>
<tr>
<td>606</td>
<td>DCOD_ERR</td>
<td>Could not handle file encoding.</td>
</tr>
<tr>
<td>607</td>
<td>RCOD_ERR</td>
<td>Recovery could not enable encoding.</td>
</tr>
<tr>
<td>608</td>
<td>IAIX_ERR</td>
<td>IIDX attributes do not match file.</td>
</tr>
<tr>
<td>609</td>
<td>LTPW_ERR</td>
<td>Temporary password failure.</td>
</tr>
<tr>
<td>610</td>
<td>HNUL_ERR</td>
<td>NULL target not permitted for this request.</td>
</tr>
<tr>
<td>611</td>
<td>HLOG_ERR</td>
<td>Could not access/find transaction log.</td>
</tr>
<tr>
<td>612</td>
<td>HSTR_ERR</td>
<td>Must make a first search call (ctHISTfirst()).</td>
</tr>
<tr>
<td>613</td>
<td>HONE_ERR</td>
<td>Can only return data OR index entries.</td>
</tr>
<tr>
<td>614</td>
<td>HMAP_ERR</td>
<td>Could not find ISAM map from specified index file to a data file.</td>
</tr>
<tr>
<td>615</td>
<td>HIDX_ERR</td>
<td>Cannot return index entries from a specified data file.</td>
</tr>
<tr>
<td>616</td>
<td>HACT_ERR</td>
<td>TransactionHistory() cannot be called during an application’s own active transaction.</td>
</tr>
<tr>
<td>617</td>
<td>HNOT_ERR</td>
<td>Did not find target.</td>
</tr>
<tr>
<td>618</td>
<td>HENT_ERR</td>
<td>No more transaction log entries.</td>
</tr>
<tr>
<td>619</td>
<td>HZRO_ERR</td>
<td>Zero recbyt not permitted on this request.</td>
</tr>
<tr>
<td>620</td>
<td>HSIZ_ERR</td>
<td>Bufsiz too small.</td>
</tr>
<tr>
<td>621</td>
<td>HTYP_ERR</td>
<td>Transaction type found in log not expected.</td>
</tr>
<tr>
<td>622</td>
<td>HMID_ERR</td>
<td>Must reset TransactionHistory() through a terminate call or preliminary log call.</td>
</tr>
<tr>
<td>623</td>
<td>HMEM_ERR</td>
<td>Not enough memory for CTHIST.</td>
</tr>
<tr>
<td>624</td>
<td>HNET_ERR</td>
<td>Net change only applies to specific match of key or record position.</td>
</tr>
<tr>
<td>625</td>
<td>HMTC_ERR</td>
<td>Must specify exactly one matching criteria: ctHISTpos or ctHISTkey or one or both of ctHISTuser and ctHISTnode.</td>
</tr>
<tr>
<td>626</td>
<td>HUND_ERR</td>
<td>Encountered an UNDTRAN (undo committed transaction) going forward. Must completely restart this set of history calls (i.e. repeat the first search call and subsequent search calls: the undone transaction will be ignored).</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>627</td>
<td>HUNK_ERR</td>
<td>Unknown type of request.</td>
</tr>
<tr>
<td>628</td>
<td>HFIL_ERR</td>
<td>Must specify <code>filno</code>.</td>
</tr>
<tr>
<td>629</td>
<td>HTFL_ERR</td>
<td>Could not initialize internal file ID: Preserve files and contact FairCom.</td>
</tr>
<tr>
<td>630</td>
<td>HUNX_ERR</td>
<td>Unexpected length in log entry.</td>
</tr>
<tr>
<td>631</td>
<td>SSLO_ERR</td>
<td>Could not get <code>ctSS_LOCK</code> on file</td>
</tr>
<tr>
<td>-632</td>
<td>LLOK_COD</td>
<td>User lost locks found on close</td>
</tr>
<tr>
<td>633</td>
<td>NPLN_ERR</td>
<td>NULL <code>plen</code> (pointer to size).</td>
</tr>
<tr>
<td>634</td>
<td>NLEN_ERR</td>
<td>Negative length specified.</td>
</tr>
<tr>
<td>635</td>
<td>TSYC_ERR</td>
<td>Could not create thread synchronization object.</td>
</tr>
<tr>
<td>636</td>
<td>TSYF_ERR</td>
<td>Thread synchronization object 'get' failed.</td>
</tr>
<tr>
<td>637</td>
<td>TSYR_ERR</td>
<td>Thread synchronization object 'rel' failed.</td>
</tr>
<tr>
<td>638</td>
<td>TQUE_ERR</td>
<td>Queue message truncated to fit.</td>
</tr>
<tr>
<td>639</td>
<td>TZRO_ERR</td>
<td>Semaphore must be initialized with count &gt; 0.</td>
</tr>
<tr>
<td>640</td>
<td>TINT_ERR</td>
<td>Semaphore already initialized.</td>
</tr>
<tr>
<td>641</td>
<td>TSYX_ERR</td>
<td>Thread synchronization object 'cls' failed.</td>
</tr>
<tr>
<td>642</td>
<td>EXCT_ERR</td>
<td>Must use exact file name.</td>
</tr>
<tr>
<td>643</td>
<td>DPND_ERR</td>
<td>Accessing a file pending delete</td>
</tr>
<tr>
<td>-644</td>
<td>RDEL_COD</td>
<td>Reversible <code>TRANDEP</code> delete</td>
</tr>
<tr>
<td>645</td>
<td>CHGF_ERR</td>
<td>File number changed after deferred close.</td>
</tr>
<tr>
<td>646</td>
<td>SDEP_ERR</td>
<td>Superfile member/host <code>TRANDEP</code> specification conflict</td>
</tr>
<tr>
<td>-647</td>
<td>E2GB_COD</td>
<td>No support beyond 2 GB</td>
</tr>
<tr>
<td>-648</td>
<td>E4GB_COD</td>
<td>No support beyond 4 GB</td>
</tr>
<tr>
<td>649</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>650</td>
<td>DUPJ_ERR</td>
<td>Duplicate keys purged and logged.</td>
</tr>
<tr>
<td>651</td>
<td>DUPX_ERR</td>
<td>Could not process duplicate key log.</td>
</tr>
<tr>
<td>652</td>
<td>DUPL_ERR</td>
<td>Duplicate keys rejected and listed.</td>
</tr>
<tr>
<td>653</td>
<td>MAPL_ERR</td>
<td>Attempt to exceed mapped lock limit.</td>
</tr>
<tr>
<td>654</td>
<td>TLNG_ERR</td>
<td>Record length too long for log size.</td>
</tr>
<tr>
<td>655</td>
<td>FREO_ERR</td>
<td>Could not reopen using <code>freopen()</code>.</td>
</tr>
<tr>
<td>656</td>
<td>LHDR_ERR</td>
<td>Transaction log header is bad.</td>
</tr>
<tr>
<td>657</td>
<td>CPYF_ERR</td>
<td>Could not create copy file.</td>
</tr>
<tr>
<td>658</td>
<td>CPYW_ERR</td>
<td>Could not write copy file.</td>
</tr>
<tr>
<td>659</td>
<td>CPYR_ERR</td>
<td>Could not read entire original file.</td>
</tr>
<tr>
<td>660</td>
<td>CPYB_ERR</td>
<td>Rebuild complete, but failed mirror copy.</td>
</tr>
<tr>
<td>661</td>
<td>CPYX_ERR</td>
<td>Failed process duplicate log and copy mirror.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>662</td>
<td>CPYJ_ERR</td>
<td>Duplicate key purged, but could not copy mirror.</td>
</tr>
<tr>
<td>663</td>
<td>CPYL_ERR</td>
<td>Duplicate key rejected, but could not copy mirror.</td>
</tr>
<tr>
<td>664</td>
<td>LPRI_ERR</td>
<td>Primary log (or start) file failed.</td>
</tr>
<tr>
<td>665</td>
<td>LMIR_ERR</td>
<td>Mirrored log (or start) file failed.</td>
</tr>
<tr>
<td>666</td>
<td>LFRM_ERR</td>
<td>Incompatible log format.</td>
</tr>
<tr>
<td>667</td>
<td>MAXZ_ERR</td>
<td>Attempt to exceed max file size.</td>
</tr>
<tr>
<td>668</td>
<td>LRGN_ERR</td>
<td>Large page size not a multiple of 16K.</td>
</tr>
<tr>
<td>669</td>
<td>XCRE_ERR</td>
<td>Inconsistent XCREblk.</td>
</tr>
<tr>
<td>670</td>
<td>HMIN_ERR</td>
<td>Node sectors too small for huge file.</td>
</tr>
<tr>
<td>671</td>
<td>XOVR_ERR</td>
<td>Non-zero high long with non-huge file.</td>
</tr>
<tr>
<td>672</td>
<td>HDR8_ERR</td>
<td>Inconsistency: file attr &amp; sys version.</td>
</tr>
<tr>
<td>673</td>
<td>SIG8_ERR</td>
<td>Extended header bad signature.</td>
</tr>
<tr>
<td>674</td>
<td>SEGM_ERR</td>
<td>Additional file segments needed.</td>
</tr>
<tr>
<td>675</td>
<td>SEGS_ERR</td>
<td>File segments not supported.</td>
</tr>
<tr>
<td>676</td>
<td>SEGO_ERR</td>
<td>Could not open segment.</td>
</tr>
<tr>
<td>677</td>
<td>SEGD_ERR</td>
<td>Cannot directly operate on seg def</td>
</tr>
<tr>
<td>678</td>
<td>SEGN_ERR</td>
<td>Bad file segment name.</td>
</tr>
<tr>
<td>679</td>
<td>SEGZ_ERR</td>
<td>Bad file segment size.</td>
</tr>
<tr>
<td>680</td>
<td>SEGC_ERR</td>
<td>Could not create file segment.</td>
</tr>
<tr>
<td>681</td>
<td>SEGH_ERR</td>
<td>Could not process segment header.</td>
</tr>
<tr>
<td>682</td>
<td>SEGL_ERR</td>
<td>Seg resource cannot move.</td>
</tr>
<tr>
<td>683</td>
<td>SEGU_ERR</td>
<td>Seg update invalid, see CTSTATUS.FCS.</td>
</tr>
<tr>
<td>684</td>
<td>SEGX_ERR</td>
<td>Segment update already in progress.</td>
</tr>
<tr>
<td>685</td>
<td>SEGF_ERR</td>
<td>I/O on segmented file terminated.</td>
</tr>
<tr>
<td>686</td>
<td>SEGR_ERR</td>
<td>Segment definition too large.</td>
</tr>
<tr>
<td>687</td>
<td>SEGQ_ERR</td>
<td>Unexpected value during recovery.</td>
</tr>
<tr>
<td>688</td>
<td>SEGP_ERR</td>
<td>Pending segment mismatch.</td>
</tr>
<tr>
<td>689</td>
<td>SEQ8_ERR</td>
<td>1st &amp; 2nd headers out of sync.</td>
</tr>
<tr>
<td>690</td>
<td>CREQ_ERR</td>
<td>Bad request header CheckSum.</td>
</tr>
<tr>
<td>691</td>
<td>CRSQ_ERR</td>
<td>Bad response header CheckSum.</td>
</tr>
<tr>
<td>692</td>
<td>CRCQ_ERR</td>
<td>Bad request (to server) CRC.</td>
</tr>
<tr>
<td>693</td>
<td>CRCP_ERR</td>
<td>Bad response (from server) CRC.</td>
</tr>
<tr>
<td>694</td>
<td>NUNC_ERR</td>
<td>No Unicode support.</td>
</tr>
<tr>
<td>695</td>
<td>BSPC_ERR</td>
<td>Could not get work buffer for blk I/O.</td>
</tr>
<tr>
<td>696</td>
<td>SEGK_ERR</td>
<td>OPENFIL() called for a segment</td>
</tr>
<tr>
<td>697</td>
<td>DSPC_ERR</td>
<td>Could not allocate encoding buffer.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>700</td>
<td>OSEG_ERR</td>
<td>Could not open key segment definition.</td>
</tr>
<tr>
<td>701</td>
<td>CSEG_ERR</td>
<td>Could not process compression options.</td>
</tr>
<tr>
<td>702</td>
<td>ASEG_ERR</td>
<td>Could not process compression attributes.</td>
</tr>
<tr>
<td>703</td>
<td>HSEG_ERR</td>
<td>Invalid key segment handle.</td>
</tr>
<tr>
<td>704</td>
<td>SSEG_ERR</td>
<td>Invalid source type.</td>
</tr>
<tr>
<td>705</td>
<td>DSEG_ERR</td>
<td>Segment definition already exists.</td>
</tr>
<tr>
<td>706</td>
<td>NSEG_ERR</td>
<td>No segment data produced.</td>
</tr>
<tr>
<td>707</td>
<td>USEG_ERR</td>
<td>No segment definition.</td>
</tr>
<tr>
<td>708</td>
<td>MBSP_ERR</td>
<td>Multi-byte names not supported.</td>
</tr>
<tr>
<td>709</td>
<td>MBNM_ERR</td>
<td>Badly formed multi-byte name.</td>
</tr>
<tr>
<td>710</td>
<td>MBFM_ERR</td>
<td>Multi-byte variant not supported.</td>
</tr>
<tr>
<td>711</td>
<td>DIDX_ERR</td>
<td>Cannot call <code>UPDCIDX()</code> while <code>DROPIDX()</code> pending.</td>
</tr>
<tr>
<td>712</td>
<td>PLOW_ERR</td>
<td>Partition number out of range - low.</td>
</tr>
<tr>
<td>713</td>
<td>PHST_ERR</td>
<td>File is not partition host.</td>
</tr>
<tr>
<td>714</td>
<td>PMBR_ERR</td>
<td>File is not partition member.</td>
</tr>
<tr>
<td>715</td>
<td>PNOT_ERR</td>
<td>Raw partition does not exist.</td>
</tr>
<tr>
<td>716</td>
<td>PXPR_ERR</td>
<td>Bad value for partition key.</td>
</tr>
<tr>
<td>717</td>
<td>POVR_ERR</td>
<td>Could not overload partition number.</td>
</tr>
<tr>
<td>718</td>
<td>PUSD_ERR</td>
<td>Partition member in use.</td>
</tr>
<tr>
<td>719</td>
<td>PPND_ERR</td>
<td>Partition member pending purge.</td>
</tr>
<tr>
<td>720</td>
<td>PPRG_ERR</td>
<td>Partition member purged.</td>
</tr>
<tr>
<td>721</td>
<td>PARC_ERR</td>
<td>Partition member archived.</td>
</tr>
<tr>
<td>722</td>
<td>PLST_ERR</td>
<td>Bad partition host list.</td>
</tr>
<tr>
<td>723</td>
<td>PTRY_ERR</td>
<td>Must retry operation.</td>
</tr>
<tr>
<td>724</td>
<td>PCRP_ERR</td>
<td>Bad current ISAM position for host.</td>
</tr>
<tr>
<td>725</td>
<td>PVRN_ERR</td>
<td><code>PARTRES</code> version error.</td>
</tr>
<tr>
<td>726</td>
<td>PPRG_COD</td>
<td>Duplicate error caused by purged part.</td>
</tr>
<tr>
<td>727</td>
<td>PFMD_ERR</td>
<td>Bad partition file mode settings.</td>
</tr>
<tr>
<td>728</td>
<td>PSUP_ERR</td>
<td>Partition support not enabled.</td>
</tr>
<tr>
<td>729</td>
<td>PUNQ_ERR</td>
<td>Purged unique global keys encountered.</td>
</tr>
<tr>
<td>730</td>
<td>PHGH_ERR</td>
<td>Partition number out of range - high.</td>
</tr>
<tr>
<td>731</td>
<td>PRIK_ERR</td>
<td>Illegal operation with primary key.</td>
</tr>
<tr>
<td>732</td>
<td>UMOD_ERR</td>
<td>Illegal file mode / <code>x8mode</code> value.</td>
</tr>
<tr>
<td>733</td>
<td>PMOP_ERR</td>
<td>Cannot open this member except read-only.</td>
</tr>
<tr>
<td>734</td>
<td>EXTH_ERR</td>
<td>Extended header required.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>735</td>
<td>CUNF_ERR</td>
<td>Client does not support UNIFRMAT server.</td>
</tr>
<tr>
<td>736</td>
<td>AOV_ERR</td>
<td>Async handle overflow.</td>
</tr>
<tr>
<td>737</td>
<td>PMTC_ERR</td>
<td>Partition member characteristics do not match.</td>
</tr>
<tr>
<td>738</td>
<td>MINT_ERR</td>
<td>ctThrdInit() must be called first.</td>
</tr>
<tr>
<td>739</td>
<td>FFLT_ERR</td>
<td>Record does not pass filter.</td>
</tr>
<tr>
<td>740</td>
<td>DDCR_ERR</td>
<td>Dynamic dump refused SF member create.</td>
</tr>
<tr>
<td>741</td>
<td>DDRR_ERR</td>
<td>Dynamic dump refused SF member delete.</td>
</tr>
<tr>
<td>742</td>
<td>S6BT_ERR</td>
<td>Superfile member idx / host 6BTRAN != 0</td>
</tr>
<tr>
<td>743</td>
<td>SFHI_ERR</td>
<td>Superfile host directory index null.</td>
</tr>
<tr>
<td>744</td>
<td>FREL_ERR</td>
<td>File requires unavailable feature.</td>
</tr>
<tr>
<td>745</td>
<td>R6BT_ERR</td>
<td>6BTRAN file required.</td>
</tr>
<tr>
<td>746</td>
<td>ACHN_ERR</td>
<td>Could not allocate I/O channel(s).</td>
</tr>
<tr>
<td>747</td>
<td>RSYN_ERR</td>
<td>Partition rule syntax error.</td>
</tr>
<tr>
<td>748</td>
<td>REXT_ERR</td>
<td>Read failed external cause: sysiocod</td>
</tr>
<tr>
<td>749</td>
<td>PBAD_ERR</td>
<td>Bad parameter value.</td>
</tr>
<tr>
<td>750</td>
<td>ICUV_ERR</td>
<td>Different ICU version, rebuild index.</td>
</tr>
<tr>
<td>751</td>
<td>CHKM_ERR</td>
<td>Checkpoint memory inconsistency.</td>
</tr>
<tr>
<td>752</td>
<td>LSET_ERR</td>
<td>More than one log set in transaction.</td>
</tr>
<tr>
<td>753</td>
<td>SPCL_ERR</td>
<td>Cannot RST/CLR past special savepoint.</td>
</tr>
<tr>
<td>754</td>
<td>QOWN_ERR</td>
<td>Only Q creator can perform operation.</td>
</tr>
<tr>
<td>755</td>
<td>SQUE_ERR</td>
<td>A system queue is required.</td>
</tr>
<tr>
<td>756</td>
<td>NACT_ERR</td>
<td>Server is not activated.</td>
</tr>
<tr>
<td>757</td>
<td>STPU_ERR</td>
<td>Must uninit c-tree (STPUSR).</td>
</tr>
<tr>
<td>758</td>
<td>QNOT_ERR</td>
<td>Only notifications to queue.</td>
</tr>
<tr>
<td>759</td>
<td>QUIN_ERR</td>
<td>Wrong queue instance.</td>
</tr>
<tr>
<td>760</td>
<td>XMON_ERR</td>
<td>SYSMON interrupted / cancelled.</td>
</tr>
<tr>
<td>761</td>
<td>NMON_ERR</td>
<td>No active SYSMON.</td>
</tr>
<tr>
<td>762</td>
<td>NRNG_ERR</td>
<td>No range defined for index.</td>
</tr>
<tr>
<td>763</td>
<td>ORNG_COD</td>
<td>Segment out of range.</td>
</tr>
<tr>
<td>764</td>
<td>CRNG_ERR</td>
<td>Range defined but no FRS/LST RNG.</td>
</tr>
<tr>
<td>765</td>
<td>CIOB_ERR</td>
<td>Comm I/O has been blocked-ctcomioblk has been set by user.</td>
</tr>
<tr>
<td>766</td>
<td>VCLS_ERR</td>
<td>Insufficient file handles and could not find file to close for virtual processing.</td>
</tr>
<tr>
<td>767</td>
<td>FALG_ERR</td>
<td>Fixed length record offset not aligned.</td>
</tr>
<tr>
<td>768</td>
<td>CMLK_ERR</td>
<td>Commit lock error: make sure record update performed with lock.</td>
</tr>
<tr>
<td>769</td>
<td>CULK_ERR</td>
<td>Unexpected CMTLOK unlock failure: Call FairCom.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>770</td>
<td>XTRN_ERR</td>
<td>Cannot turn off file's tran support in middle of transaction if file updated.</td>
</tr>
<tr>
<td>771</td>
<td>COMP_ERR</td>
<td>Compatibility option not enabled.</td>
</tr>
<tr>
<td>772</td>
<td>PLAT_ERR</td>
<td>Platform does not support compatibility option.</td>
</tr>
<tr>
<td>773</td>
<td>SREP_ERR</td>
<td>Superfile host &amp; member must have same <code>REPLICATION</code> attribute.</td>
</tr>
<tr>
<td>774</td>
<td>SMEM_ERR</td>
<td>Superfile members must all be closed.</td>
</tr>
<tr>
<td>775</td>
<td>UNQK_ERR</td>
<td>No UNQKEY support for <code>REPLICATION</code>. If a low-level file open call that requests write access to a c-tree data file fails with this error, it is because the data file has replication enabled. A c-tree data file that has replication enabled can only be opened for write access at the ISAM level. A read-only low-level call is allowed: use the `ctREADFIL</td>
</tr>
<tr>
<td>776</td>
<td>SUPR_ERR</td>
<td>Operation not supported for <code>SUPERFILES</code>.</td>
</tr>
<tr>
<td>777</td>
<td>REPU_ERR</td>
<td>Cannot unregister another client's replication instance.</td>
</tr>
<tr>
<td>778</td>
<td>REPI_ERR</td>
<td>The specified replication instance name is not registered.</td>
</tr>
<tr>
<td>779</td>
<td>REPR_ERR</td>
<td>The specified replication instance name is already registered.</td>
</tr>
<tr>
<td>780</td>
<td>REPA_ERR</td>
<td>Cannot attach to replication instance with active connection.</td>
</tr>
<tr>
<td>781</td>
<td>REPC_ERR</td>
<td>This replication connection already has a registered instance name.</td>
</tr>
<tr>
<td>782</td>
<td>UCMP_ERR</td>
<td>Decompression error: unexpected output length.</td>
</tr>
<tr>
<td>783</td>
<td>IITI_ERR</td>
<td>Incremental index: cannot add permanent indices while temporary indices exist.</td>
</tr>
<tr>
<td>784</td>
<td>LDPI_ERR</td>
<td>LDAP initialization failed.</td>
</tr>
<tr>
<td>785</td>
<td>LDPC_ERR</td>
<td>Error connecting to LDAP server.</td>
</tr>
<tr>
<td>786</td>
<td>LDPB_ERR</td>
<td>Error binding to LDAP server.</td>
</tr>
<tr>
<td>787</td>
<td>LDPA_ERR</td>
<td>LDAP user authentication failed.</td>
</tr>
<tr>
<td>788</td>
<td>LDPG_ERR</td>
<td>Error checking user's LDAP groups.</td>
</tr>
<tr>
<td>789</td>
<td>LGRP_ERR</td>
<td>Member of c-tree login group required.</td>
</tr>
<tr>
<td>790</td>
<td>LSST_ERR</td>
<td>Strict serializer must be in transaction to access record.</td>
</tr>
<tr>
<td>791</td>
<td>LSSK_ERR</td>
<td>Strict serializer cannot keep locks.</td>
</tr>
<tr>
<td>792</td>
<td>XSHT_ERR</td>
<td>External server shutdown disabled. If replication is running on a server, it should be stopped before trying to stop the server.</td>
</tr>
<tr>
<td>793</td>
<td>PKSP_ERR</td>
<td>Partial record rewrite has keys spanning partial record and remaining region of record.</td>
</tr>
<tr>
<td>794</td>
<td>XBUF_ERR</td>
<td>Tried to update data with missing key buffer contents.</td>
</tr>
<tr>
<td>795</td>
<td>VTSM_COD</td>
<td>Recursive <code>ctptsema()</code> call for open or create.</td>
</tr>
<tr>
<td>796</td>
<td>FCPY_ERR</td>
<td>File copy failed.</td>
</tr>
</tbody>
</table>
## c-treeDB C++ API Class Reference

### Value | Symbolic Constant | Explanation
--- | --- | ---
797 | DRST_ERR | Immediate dump restore failed.
798 | FBLK_ERR | File is blocked, retry later.
799 | FBLK_RDO | File block cleared: close/reopen file.

### Value | Symbolic Constant | Explanation
--- | --- | ---
800 | TDEP_ERR | TRANDEP file operation pending
801 | FBLK_PND | trying to set file block. leave core.
802 | FBLK_SUP | file block not supported for file type
803 | LNEW_ERR | existing lock not replaced: cts_llok81()
804 | SETO_ERR | cannot override configuration option that was specified in settings file
805 | MSTK_ERR | no memory for func stack alloc
806 | KCON_ERR | failed to connect to kernel engine
807 | MSTK_COD | ctDBGstack limit exceeded: debugging only
808 | TRQS_ERR | request timed out
809 | TRSP_ERR | response timed out
810 | SLOG_ERR | status log write failure
811 | IAPI_ERR | update only from internal API
812 | ZRCZ_ERR | unexpected zero node size in header
813 | UBLK_ERR | unexpected state for user block obj
814 | FBLK_ACT | thread did not have ctFBActive set
815 | FBLK_NTF | ctFILBLK already in progress for file
816 | QTUQ_ERR | only one ctQUIET process at a time
817 | QTAB_ERR | transaction aborted by ctQUIET
818 | QTFB_ERR | ctQUIET / ctFILBLK conflict
819 | QTBB_ERR | unexpected failure to block thread
820 | QTBK_PND | trying to get QUIET. leave core.
821 | MLAB_ERR | transaction abandoned: MAX_USER_LOGS
822 | MLHG_ERR | abort request would be suspended
823 | TRAB_COD | QERR or MLAB_ERR but operation performed
824 | MIMP_ERR | problem impersonating thread
825 | QTOP_ERR | ctQUIET called with files opened
826 | QBAD_ERR | improper ctQT actions: see sysiocod:
  - QBKU_COD: 1 - block and unblock actions mixed
  - QBUN_COD: 2 - cannot block after unblock has begun
  - QMBK_COD: 3 - mixed file block types
  - QSPC_COD: 4 - missing filespec
<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QMAP_COD: 5</td>
<td>mixed API block types</td>
</tr>
<tr>
<td></td>
<td>QBAP_COD: 6</td>
<td>blockAllFiles requires blockAPI</td>
</tr>
<tr>
<td></td>
<td>QBFA_COD: 7</td>
<td>cannot BlockAllFiles after blockFiles</td>
</tr>
<tr>
<td></td>
<td>QFAP_COD: 8</td>
<td>file flush requires blockAPI</td>
</tr>
<tr>
<td></td>
<td>QLAP_COD: 9</td>
<td>blockLogFiles requires blockAPI</td>
</tr>
<tr>
<td></td>
<td>QDUP_COD: 10</td>
<td>block action already requested</td>
</tr>
<tr>
<td></td>
<td>QLOG_COD: 11</td>
<td>unblockLogFiles required</td>
</tr>
<tr>
<td></td>
<td>QLGR_COD: 12</td>
<td>cannot mix ctQTlog_restorepoint with other block modes</td>
</tr>
<tr>
<td></td>
<td>QLGK_COD: 13</td>
<td>ctQT_logrestorepoint can only be used with ctQT_logrestorepoint</td>
</tr>
<tr>
<td>827</td>
<td>UTIM_OUT</td>
<td>User block timed out.</td>
</tr>
<tr>
<td>-828</td>
<td>ICUV_COD</td>
<td>ICU version updated.</td>
</tr>
<tr>
<td>-829</td>
<td>ICUV_REB</td>
<td>ICU version updated &amp; rebuild required.</td>
</tr>
<tr>
<td>830</td>
<td>TTAB_ERR</td>
<td>Transaction abandoned: TRAN_TIMEOUT.</td>
</tr>
<tr>
<td>831</td>
<td>TTHG_ERR</td>
<td>Abort request would be suspended.</td>
</tr>
<tr>
<td>-832</td>
<td>ITMP_COD</td>
<td>sysiocod value when ITIM_ERR occurs on a temporary index and record is skipped.</td>
</tr>
<tr>
<td>833</td>
<td>XNOD_ERR</td>
<td>The connection attempt has been rejected because it would exceed the maximum number of concurrent client machines allowed.</td>
</tr>
<tr>
<td>834</td>
<td>XCON_ERR</td>
<td>The connection attempt has been rejected because it would exceed the maximum number of concurrent connections allowed from this client machine.</td>
</tr>
<tr>
<td>835</td>
<td>TRAB_ERR</td>
<td>Transaction aborted (e.g., MLAB_ERR or TTAB_ERR) before the requested operation was processed. (Compare with TRAB_COD.)</td>
</tr>
<tr>
<td>836</td>
<td>LCON_ERR</td>
<td>The connection attempt has been rejected because only connections from the local system are allowed.</td>
</tr>
<tr>
<td>-837</td>
<td>IDUP_COD</td>
<td>sysiocod value when KDUP_ERR occurs on a temporary index and error is ignored.</td>
</tr>
<tr>
<td>838</td>
<td>CEXC_ERR</td>
<td>clnleaf(CLNIDXX) failed to clean node. Very unexpected; call FairCom.</td>
</tr>
<tr>
<td>839</td>
<td>CTRN_ERR</td>
<td>Index file requires key level lock cleaning.</td>
</tr>
<tr>
<td>-840</td>
<td>CTRN_COD</td>
<td>sysiocod when read only, admin open request blocked by on the fly CLNIDXX.</td>
</tr>
<tr>
<td>841</td>
<td>SHMC_ERR</td>
<td>A connection attempt using the shared memory protocol failed due to an incompatibility between client and server.</td>
</tr>
<tr>
<td>842</td>
<td>FBLK_ABT</td>
<td>Could not clear all threads from core, abort ctFILBLK() attempt.</td>
</tr>
<tr>
<td>843</td>
<td>QTBK_ABT</td>
<td>Could not clear all threads from core, abort ctQUIET() attempt.</td>
</tr>
<tr>
<td>-844</td>
<td>BTNO_COD</td>
<td>Batch has been closed.</td>
</tr>
<tr>
<td>845</td>
<td>DNCT_ERR</td>
<td>Index does not have distinct attribute.</td>
</tr>
<tr>
<td>846</td>
<td>DNCT_XCL</td>
<td>DISTINCT set update in progress.</td>
</tr>
<tr>
<td>847</td>
<td>RPTD_ERR</td>
<td>Auto recovery interrupted.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>-848</td>
<td>RDND_COD</td>
<td>sysiod: LOKDYN demotion denied</td>
</tr>
<tr>
<td>-849</td>
<td>SAVP_COD</td>
<td>sysiod: LOKDYN free denied</td>
</tr>
<tr>
<td>850</td>
<td>TR_CLIL_ERR</td>
<td>Transactional replication: Failed to start c-tree remote client subsystem. See CTSTATUS.FCS for details.</td>
</tr>
<tr>
<td>851</td>
<td>TR_RMAP_ERR</td>
<td>Transactional replication: Failed to open replication mapping file. See CTSTATUS.FCS for details.</td>
</tr>
<tr>
<td>852</td>
<td>TR_RATR_ERR</td>
<td>Transactional replication: Failed to open replica for writing because the file does not meet the requirements for replication. See CTSTATUS.FCS for details.</td>
</tr>
<tr>
<td>853</td>
<td>TR_FNOD_ERR</td>
<td>Transactional replication: Failed to set node name for connection to master server. See CTSTATUS.FCS for details.</td>
</tr>
<tr>
<td>854</td>
<td>TR_REDF_ERR</td>
<td>Transactional replication: Attempted to update replica without enabling transactional replication for the connection.</td>
</tr>
<tr>
<td>855</td>
<td>TR_SAVP_ERR</td>
<td>Transactional replication: Failed to establish a savepoint because savepoint number is out of sync with master.</td>
</tr>
<tr>
<td>856</td>
<td>TR_RDIF_ERR</td>
<td>Transactional replication: Failed to read record for update. Local record differs from master record.</td>
</tr>
<tr>
<td>857</td>
<td>XSUP_ERR</td>
<td>In multi-user standalone mode: The superfile host is open in exclusive mode. The superfile member must be opened in exclusive mode, not shared mode.</td>
</tr>
<tr>
<td>858</td>
<td>TR_NSUP_ERR</td>
<td>Transactional replication: This feature is not supported for replicas.</td>
</tr>
<tr>
<td>859</td>
<td>UVRC_ERR</td>
<td>The client's structure definition for the file FAIRCOM.FCSIUSER.dat is out of date. Update client library.</td>
</tr>
<tr>
<td>860</td>
<td>UVRS_ERR</td>
<td>The server's structure definition for the file FAIRCOM.FCSIUSER.dat is out of date. Update c-tree Server.</td>
</tr>
<tr>
<td>861</td>
<td>KLLX_ERR</td>
<td>After recovery, a key-level lock for an undone transaction discovered in optional diagnostic scan.</td>
</tr>
<tr>
<td>862</td>
<td>UNPG_ERR</td>
<td>LOCK_CACHE: System pagesize query failed.</td>
</tr>
<tr>
<td>863</td>
<td>IMPD_ERR</td>
<td>The request to impersonate the specified connection was denied because the target connection does not allow impersonation.</td>
</tr>
<tr>
<td>864</td>
<td>IMPU_ERR</td>
<td>The request to impersonate the specified connection was denied because the target connection does not allow impersonation by the specified connection.</td>
</tr>
<tr>
<td>865</td>
<td>IMPA_ERR</td>
<td>The request to impersonate the specified connection was denied because the target connection is already being impersonated.</td>
</tr>
<tr>
<td>866</td>
<td>IMPB_ERR</td>
<td>The request to impersonate the specified connection was denied because the target connection is executing a database operation or is blocked.</td>
</tr>
<tr>
<td>867</td>
<td>CBKD_ERR</td>
<td>Failed to load the filter callback library. See CTSTATUS.FCS for details.</td>
</tr>
<tr>
<td>868</td>
<td>CBKF_ERR</td>
<td>Failed to resolve the filter callback function in the filter callback DLL. See CTSTATUS.FCS for details.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>869</td>
<td>RCBK_ERR</td>
<td>A user-defined callback function terminated the rebuild operation.</td>
</tr>
<tr>
<td>870</td>
<td>CBKV_ERR</td>
<td>The filter callback DLL version is not compatible with the c-tree Server's filter callback version. See CTSTATUS.FCS for details.</td>
</tr>
<tr>
<td>871</td>
<td>XFR_SOPN_ERR</td>
<td>The file transfer operation failed because the source file could not be opened for reading. Check sysiocod for the system error code.</td>
</tr>
<tr>
<td>872</td>
<td>XFR_DOPN_ERR</td>
<td>The file transfer operation failed because the destination file could not be written. Check sysiocod for the system error code.</td>
</tr>
<tr>
<td>873</td>
<td>XFR_READ_ERR</td>
<td>The file transfer operation failed because the source file could not be read. Check sysiocod for the system error code.</td>
</tr>
<tr>
<td>874</td>
<td>XFR_WRITE_ERR</td>
<td>The file transfer operation failed because the destination file could not be written. Check sysiocod for the system error code.</td>
</tr>
<tr>
<td>875</td>
<td>XFR_BCON_ERR</td>
<td>A bound database connection called the file transfer function, but this function is supported for client connections only.</td>
</tr>
<tr>
<td>876</td>
<td>XFR_BSIZ_ERR</td>
<td>The file transfer operation failed because the caller specified an invalid file transfer block size.</td>
</tr>
<tr>
<td>877</td>
<td>XFR_SFNM_ERR</td>
<td>The file transfer operation failed because the caller specified a NULL or empty source file name.</td>
</tr>
<tr>
<td>878</td>
<td>XFR_DFNM_ERR</td>
<td>The file transfer operation failed because the caller specified a NULL or empty destination file name.</td>
</tr>
<tr>
<td>879</td>
<td>XFR_VER_ERR</td>
<td>The version of the file transfer structure supplied by the caller is not compatible with the c-tree library's structure definition. Check sysiocod for the required file transfer structure version.</td>
</tr>
<tr>
<td>880</td>
<td>XFR_DEXS_ERR</td>
<td>The file transfer operation failed because the destination file exists and the caller did not specify that the destination file is to be overwritten.</td>
</tr>
<tr>
<td>881</td>
<td>XFR_TREP_ERR</td>
<td>The file transfer operation between a local and master server failed because the server does not support the transactional replication feature.</td>
</tr>
<tr>
<td>882</td>
<td>XFR_TRLC_ERR</td>
<td>The file transfer operation between a local and master server failed because the server is not configured as a local server. Use the REPL_MAPPINGS option to configure the server as a local server.</td>
</tr>
<tr>
<td>883</td>
<td>MUOP_ERR</td>
<td>An attempt was made to open a file multiple times by the same user with different user file numbers but this support is disabled.</td>
</tr>
<tr>
<td>-884</td>
<td>DUFL_COD</td>
<td>A sysiocod value that indicates a user file number mismatch during lock or unlock operations.</td>
</tr>
<tr>
<td>-885</td>
<td>OCHK_COD</td>
<td>A sysiocod value that indicates this API call to perform a checkpoint took no action because a checkpoint was already in progress.</td>
</tr>
<tr>
<td>886</td>
<td>UFLK_ERR</td>
<td>Unexpected CHG_UFLOCK failure. Call FairCom (sysiocod specifies locale).</td>
</tr>
<tr>
<td>887</td>
<td>DUFL_ERR</td>
<td>Promoting a secondary lock from read to write is not supported.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>888</td>
<td>CONGRP_INVITAS</td>
<td>The specified task ID does not correspond to an active client connection.</td>
</tr>
<tr>
<td>889</td>
<td>CONGRP_INUSE_ERR</td>
<td>The specified connection is a member of a connection group.</td>
</tr>
<tr>
<td>890</td>
<td>HSTNAM_ERR</td>
<td>An attempt to get the name of the host system failed. Check that the system's host name is properly configured.</td>
</tr>
<tr>
<td>891</td>
<td>HSTADR_ERR</td>
<td>An attempt to get the IP addresses associated with the host system failed. Check that TCP/IP name resolution is properly configured on the host system.</td>
</tr>
<tr>
<td>892</td>
<td>MBUF_ERR</td>
<td>An attempt to perform a sequential, index-based retrieval when the index does not have a current key value. This may occur because cISAMKBUFhdr has turned off key buffer updates (MBUF_ERR); or a partial record read or changing current ISAM record location does not permit the current key value to be assembled (DBUF_ERR); or a conditional index constraint or NUL key results in no key value (NKEY_ERR).</td>
</tr>
<tr>
<td>893</td>
<td>DBUF_ERR</td>
<td></td>
</tr>
<tr>
<td>894</td>
<td>NKEY_ERR</td>
<td></td>
</tr>
<tr>
<td>895</td>
<td>TSEG_ERR</td>
<td>Partition index segment definition does not satisfy the following constraint: there must be at least one segment used to perform the key-to-partition map, and all the segments used must come at the beginning of the key.</td>
</tr>
<tr>
<td>896</td>
<td>VSEG_ERR</td>
<td>Covering index segment definitions do not map to partition index. Call FairCom.</td>
</tr>
<tr>
<td>897</td>
<td>KINC_ERR</td>
<td>Covering index segment map (ptmap) is empty.</td>
</tr>
<tr>
<td>898</td>
<td>QABN_ERR</td>
<td>A logon fails with QABN_ERR when a ctQUIET call has been abandoned, blocking attempts to validate user logons. Only the super user can logon when a ctQUIET has been abandoned. The super user can then call ctQUIET with a ctQTunblockALL argument.</td>
</tr>
<tr>
<td>899</td>
<td>PMCN_ERR</td>
<td>Cannot perform ISAM Context operations on partition member unless it has been opened in stand-alone mode (i.e., independently of the host file).</td>
</tr>
<tr>
<td>900</td>
<td>SEQDUP_ERR</td>
<td>A sequence having the specified name already exists.</td>
</tr>
<tr>
<td>901</td>
<td>SEQNAME_ERR</td>
<td>Create Sequence: An invalid sequence name was specified: the name is NULL, empty, or too long. Open Sequence: No sequence exists that has the specified name.</td>
</tr>
<tr>
<td>902</td>
<td>SEQHND_ERR</td>
<td>The specified sequence handle is invalid.</td>
</tr>
<tr>
<td>903</td>
<td>SEQTYE_ERR</td>
<td>The specified sequence type contains an invalid combination of sequence type options. The sequence type must be set to either ctSEQINC (incrementing sequence) or ctSEQDEC (decrementing sequence), and either ctSEQCYC (cycling sequence) or ctSEQTRM (terminating sequence), and optionally includes ctSEQLIM (enforce sequence limit).</td>
</tr>
<tr>
<td>904</td>
<td>SEQINI_ERR</td>
<td>The initial value specified for the sequence is out of range. If the sequence enforces a limit, the initial sequence value must be less than the sequence limit for an incrementing sequence or must be greater than the sequence limit for a decrementing sequence.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>905</td>
<td>SEQCUR_ERR</td>
<td>The current value specified for the sequence is out of range. For an incrementing sequence, the current sequence value must be greater than or equal to the initial sequence value, and if the sequence enforces a limit the current sequence value must be less than or equal to the limit value. For a decrementing sequence, the current sequence value must be less than or equal to the initial sequence value, and if the sequence enforces a limit the current sequence value must be greater than or equal to the sequence limit.</td>
</tr>
<tr>
<td>906</td>
<td>SEQLIM_ERR</td>
<td>The limit value specified for the sequence is out of range. The sequence limit must be greater than the initial sequence value for an incrementing sequence or must be less than the initial sequence value for a decrementing sequence. The sequence limit is only enforced if the sequence type specifies the ctSEQLIM bit.</td>
</tr>
<tr>
<td>907</td>
<td>SEQINC_ERR</td>
<td>The increment value specified for the sequence is out of range. The sequence increment amount must be a positive value that is less than the difference between the initial sequence value and the sequence limit.</td>
</tr>
<tr>
<td>908</td>
<td>DSTX_ERR</td>
<td>The file header indicates that the maximum number of partial key distinct counts supported by the file is greater than the system support limit: ctMAXSEGdistinct. The file cannot be opened.</td>
</tr>
<tr>
<td>909</td>
<td>VFYVER_ERR</td>
<td>The structure version specified for the input and output structures is not supported by this version of the code.</td>
</tr>
<tr>
<td>910</td>
<td>VFYTRM_ERR</td>
<td>A user-defined callback function terminated the index verify operation.</td>
</tr>
<tr>
<td>911</td>
<td>NSUP_DAR</td>
<td>File contains a Direct Access Resource (DAR) that is not supported. sysiocod contains the type of DAR. See ctopt2.h for dartyp assignments.</td>
</tr>
<tr>
<td>912</td>
<td>URES_ERR</td>
<td>Special resource can only be added by system routine. sysiocod holds the special resource type.</td>
</tr>
<tr>
<td>913</td>
<td>DDAR_ERR</td>
<td>Duplicate DAR not allowed. sysiocod holds the 'dartyp' value. See FC_DAR_TYPES in ctopt2.h for interpretation.</td>
</tr>
<tr>
<td>914</td>
<td>XDAR_ERR</td>
<td>Requested DAR does not exist.</td>
</tr>
<tr>
<td>915</td>
<td>LAUG_ERR</td>
<td>Log file requires augmented log entries that are not supported.</td>
</tr>
<tr>
<td>916</td>
<td>NDAR_ERR</td>
<td>Mismatch between header DAR count, and DARs found in resource chain.</td>
</tr>
<tr>
<td>917</td>
<td>ODAR_ERR</td>
<td>Reached limit on the number of instances of a particular DAR type for a single file. Cannot add any more.</td>
</tr>
<tr>
<td>918</td>
<td>RDAR_ERR</td>
<td>DAR attribute word is marked deleted (FC_DAR_delete bit). A nonTran file DAR delete forces DAR image to disk with FC_DAR_delete bit set in attribute word.</td>
</tr>
<tr>
<td>919</td>
<td>REPL_ERR</td>
<td>Low-level operations are not allowed on replicated files.</td>
</tr>
<tr>
<td>920</td>
<td>FNAC_ERR</td>
<td>The file exists but could not be accessed. Check the system error code stored in sysiocod for the reason.</td>
</tr>
<tr>
<td>921</td>
<td>IDFL_CHG</td>
<td>Attempt to change IDfield during an ISAM rewrite operation. Rewrite fails.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>922</td>
<td>IDFL_BUF</td>
<td>Attempt an ISAM rewrite operation without current ISAM IDfield buffer contents. No way to determine if IDfield has been changed. Rewrite fails.</td>
</tr>
<tr>
<td>923</td>
<td>IERR_COD</td>
<td>An internal error has occurred; similar to a catastrophic terr. Instead of terminating execution, IERR_COD is returned. sysiocod is set to a local code, and a message is written into CTSTATUS.FCS.</td>
</tr>
<tr>
<td>924</td>
<td>SADM_ERR</td>
<td>Only the super administrator user account (named ADMIN) can perform this operation. For example, only ADMIN can change ADMIN group membership.</td>
</tr>
<tr>
<td>925</td>
<td>SCMP_ERR</td>
<td>The c-tree client is attempting to use features of the SECURITY API function that this c-tree Server does not support. Update your c-tree Server.</td>
</tr>
<tr>
<td>926</td>
<td>HLMT_ERR</td>
<td>The transaction history log scan terminated because the user-specified limit on the number of logs to scan was reached.</td>
</tr>
<tr>
<td>927</td>
<td>PNST_ERR</td>
<td>The instance number of a partition does not match the instance number maintained for the partition by the host file. Typically the instance number of a partition is increased only when it has been purged and then a new partition (with the same raw partition number) is created; or an individual partition member has been rebuilt through a call to PTADMIN.</td>
</tr>
<tr>
<td>928</td>
<td>PMRB_ERR</td>
<td>Partitioned file rebuild did not successfully rebuild all partitions. See CTSTATUS.FCS for more details.</td>
</tr>
<tr>
<td>929</td>
<td>PRBL_ERR</td>
<td>Partitioned file rebuild requires special support. Rebuilding the entire partitioned file requires ctPARTITION to be defined at compile time. Rebuilding an individual partition member may require ctPARTITIONinstnc to be defined at compile time, and the member rebuild must be initiated through PTADMIN().</td>
</tr>
<tr>
<td>930</td>
<td>PNSX_ERR</td>
<td>Partition member instance # exceeds maximum of 255 for a non-huge partitioned file. Use RBLIFILX8 to rebuild the entire set of partitioned files.</td>
</tr>
<tr>
<td>931</td>
<td>PALL_ERR</td>
<td>Could not rebuild only bad partitioned file components. Must rebuild entire partitioned file. Remove badpartIFIL from tfilno member of IFIL used in RBLIFILX8.</td>
</tr>
<tr>
<td>932</td>
<td>BMPW_ERR</td>
<td>The specified encryption master password is incorrect.</td>
</tr>
<tr>
<td>933</td>
<td>ICOD_ERR</td>
<td>An encryption operation failed due to an unexpected internal error. See CTSTATUS.FCS for details.</td>
</tr>
<tr>
<td>934</td>
<td>IICT_ERR</td>
<td>Encountered IICT operations without ctFeatIICT enabled. (IICT stands for Immediate Independent Commit Transaction: a single ISAM update operation treated as a separate transaction within an existing, pending transaction.)</td>
</tr>
<tr>
<td>935</td>
<td>IICT_FIL</td>
<td>File has already been updated within a transaction. Open the file with a different user file number and set IICT state with the new file number. Or, an add/rebuild index called for file previously placed in IICT state.</td>
</tr>
<tr>
<td>936</td>
<td>AREC_BUF</td>
<td>Could not allocate a buffer for an augmented variable length record operation (e.g., ctCMPREC).</td>
</tr>
<tr>
<td>937</td>
<td>AREC_SUP</td>
<td>Augmented record support not available (ctAUGREC / ctCMPREC / ctFeatFLEXREC).</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>938</td>
<td>AREC_DCM</td>
<td>Could not decompress the data record. sysiocod set to Z_xyz_ERROR.</td>
</tr>
<tr>
<td>939</td>
<td>AREC_ISM</td>
<td>Compressed record length files require ISAM update; not low-level update.</td>
</tr>
<tr>
<td>940</td>
<td>AREC_NOP</td>
<td>Requested operation not supported for augmented records.</td>
</tr>
<tr>
<td>941-942</td>
<td></td>
<td>See 941-942, 960-962 Logon Error Codes.</td>
</tr>
<tr>
<td>943</td>
<td>TFLN_ERR</td>
<td>c-tree does not support a rebuild or compact option that you specified in the IFIL’s tfilno field.</td>
</tr>
<tr>
<td>944</td>
<td>AREC_MOD</td>
<td>Bad augmented record file mode at create. For example, specifying ctAugmentedFxd without ctVLENGTH. Two of the conditions that return this error are: • Setting ctCompressRec in the XCREblk corresponding to an index file. It is only allowed for data files. • Setting ctCompressRec on a data file that is not variable-length (ctVLENGTH).</td>
</tr>
<tr>
<td>945</td>
<td>AREC_FXD</td>
<td>Add/update varlen must match fixed length for file with ctAugmentedFxd specified at create.</td>
</tr>
</tbody>
</table>
| 946   | CMPR_ERR          | Error setting up compression routines. Check sysiocod for details: sysiocod Explanation 
1 Non-zlib compression requires DLL 
2 Bad version number 
3 Inconsistent attrstr/attrlen 
4 DLL name Unicode conversion error 
5 DLL load error 
6->13 Function name Unicode conversion error 
14->21 Failed to resolve function proc address 
32 Type/version mismatch in FncInit() 
33 Could not allocate attrval in FncInit() 
34 attrlen disagreement in FncInit() 
35 Could not initialize compress context in CmpInit() 
36 Could not initialize decompress context in ExpInit() 
37 Invalid attrstr contents found by FncInit() 
38 Could not allocate dedicated attrstr in ctSETCOMPRESS 
1000+err where err is a DLL-specific unexpected error code 
2000+err where err is a DLL-specific code related to corrupted |
<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>947</td>
<td>SMLO_ERR</td>
<td>An internal error code returned by the shared memory communication protocol receive function for Unix systems when the receive for a logoff operation fails on the semaphore wait. This is a normal occurrence if the c-tree Server has removed the semaphore set before the client read the response. In this case the client side code returns success.</td>
</tr>
<tr>
<td>948</td>
<td>PMXS_ERR</td>
<td>Operation on partition host cannot be performed if partition members already exist. Operation must be performed after the host is created, but before any partition members are created.</td>
</tr>
<tr>
<td>949</td>
<td>IEXS_ERR</td>
<td>The request to start an IICT failed because an IICT is already active.</td>
</tr>
<tr>
<td>950</td>
<td>INON_ERR</td>
<td>The request to end an IICT failed because no IICT is active.</td>
</tr>
<tr>
<td>951</td>
<td>IACT_ERR</td>
<td>The request to commit the transaction failed because an IICTxtd is active. First commit or abort the IICTxtd. You may also abort the transaction, which first aborts the IICTxtd.</td>
</tr>
<tr>
<td>952</td>
<td>ENCK_ERR</td>
<td>The current master encryption key's attributes are inconsistent with the master key used to encrypt the security resource of the specified file. For example, you might be using a 128-bit master key and are trying to read a file whose security resource was encrypted with a 256-bit master key.</td>
</tr>
<tr>
<td>953</td>
<td>QTOC_ERR</td>
<td>A call to suspend a thread because of ctQUIET processing failed because the thread owned the file open/close semaphore.</td>
</tr>
<tr>
<td>954</td>
<td>NCMP_ERR</td>
<td>Compacting a partitioned file is not supported.</td>
</tr>
<tr>
<td>955</td>
<td>DOTX_ERR</td>
<td>Attempt to acquire internal mutex, ctpdotsema, out of order. Unexpected error, contact FairCom.</td>
</tr>
<tr>
<td>956</td>
<td>APND_ERR</td>
<td>Partition member pending archive</td>
</tr>
<tr>
<td>957</td>
<td>DMAP_ERR</td>
<td>This index file is already mapped to a different data file.</td>
</tr>
<tr>
<td>958</td>
<td>IMIS_ERR</td>
<td>Data record is missing identity field value.</td>
</tr>
<tr>
<td>959</td>
<td>LSEV_ERR</td>
<td>This c-tree client uses a different secure logon version than your c-tree Server. Update your c-tree client.</td>
</tr>
<tr>
<td>960-962</td>
<td>See 941-942, 960-967 Logon Error Codes.</td>
<td></td>
</tr>
<tr>
<td>963</td>
<td>LISM_ERR</td>
<td>An attempt to close an ISAM data file with the low-level file close routine CLSFIL. COMPATIBILITY CLSFIL_ISAM restores the old behavior when CLSFIL is called for an ISAM data file. For standalone applications, the CLSFIL behavior can be restored by compiling the code with NO_ctBEHAV_CLSFIL_ISAM.</td>
</tr>
<tr>
<td>964</td>
<td>SCNT_ERR</td>
<td>Pre-image space overflow of image update count. Each update of a record location, within a single transaction, causes the pre-image update count to be incremented. Repeated update of the same record within the transaction eventually overflows the update count.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>965</td>
<td>BCOD_ERR</td>
<td>c-tree Server was not able to decrypt the buffer sent to it by the client. This is an unexpected error. Contact FairCom support.</td>
</tr>
<tr>
<td>966</td>
<td>BORD_ERR</td>
<td>c-tree was compiled with the wrong byte order for this platform.</td>
</tr>
<tr>
<td>967</td>
<td>ULMT_ERR</td>
<td>See 941-942, 960-967 Logon Error Codes.</td>
</tr>
<tr>
<td>968</td>
<td>GLMT_ERR</td>
<td>Logon is denied because one of the groups for this user account has reached its maximum number of concurrent logons.</td>
</tr>
<tr>
<td>969</td>
<td>QTBK_THD</td>
<td>Attempt to suspend the thread that invoked <code>ctQUIET</code>. Such a suspension would hang the server. <code>sysiocod</code> is set to the type of suspension request such as <code>ctQTtranbeg</code>; or to a negative <code>cterrc.h</code> entry such as <code>LWRT_COD</code>.</td>
</tr>
<tr>
<td>970</td>
<td>ISAM_MUSR_ERR</td>
<td>The connection attempt has been rejected because it would exceed the maximum number of concurrent ISAM connections allowed.</td>
</tr>
<tr>
<td>971</td>
<td>SQL_MUSR_ERR</td>
<td>The connection attempt has been rejected because it would exceed the maximum number of concurrent SQL connections allowed.</td>
</tr>
<tr>
<td>-972</td>
<td>PRTL_COD</td>
<td><code>sysiocod</code> is set to PRTL_COD if rebuild detects a partial record (or resource) at the end of a truncated data file. If the purgeIFIL option is used, a copy of the partial record is placed in a temporary file; the record is overwritten with 0xff bytes; and <code>sysiocod</code> is set to PRTL_FF.</td>
</tr>
<tr>
<td>-973</td>
<td>PRTL_FF</td>
<td>If the record is successfully dumped to the temporary file, but there is a problem overwriting the partial record with 0xff bytes, <code>sysiocod</code> will be set to PRTL_COD not PRTL_FF. Whether or not the purge option is used, <code>CTSTATUS.FCS</code> contains the byte offset and length of the partial record. (Prior to V10.3, the values of PRTL_COD and PRTL_FF were -970 and -971 respectively.)</td>
</tr>
<tr>
<td>-974</td>
<td>LWRT_COD</td>
<td>Attempt to write to log files when <code>ctQTblockLogFiles</code> is in force. (Prior to V10.3, the value of this error code was -972)</td>
</tr>
<tr>
<td>-975</td>
<td>FNOP_COD</td>
<td><code>sysiocod</code> value to indicate to caller of file open that the file is not already open. (Prior to V10.3, the value of this error code was -10)</td>
</tr>
<tr>
<td>-976</td>
<td>FNOR_COD</td>
<td><code>sysiocod</code> value to indicate to the caller of <code>ctrcvopn81()</code> that FNOP_ERR occurred because the file is not on the dump recovery list. (Prior to V10.3, the value of this error code was -11)</td>
</tr>
<tr>
<td>977</td>
<td>XATO_ERR</td>
<td>File create called with a <code>segmax &gt; 1</code> but without <code>ctSEGAUTO</code> bit set in <code>x8mode</code>. <code>segmax</code> and <code>x8mode</code> are members of XCREblk.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>978</td>
<td>SHMP_ERR</td>
<td>A shared memory connection could not be established because the system denied access to the client process. For example, if a client is run as a Windows service and c-tree Server is not run as a Windows service (or vice-versa), Windows does not allow c-tree Server the access to the client process which is needed to establish the shared memory connection.</td>
</tr>
<tr>
<td>979</td>
<td>FLIC_ERR</td>
<td>c-tree is not licensed to use this feature.</td>
</tr>
<tr>
<td>980</td>
<td>KEYX_ERR</td>
<td>Secure key exchange failed. See CTSTATUS.FCS for details.</td>
</tr>
<tr>
<td>981</td>
<td>CDLL_ERR</td>
<td>Could not load the specified DLL or shared library. See CTSTATUS.FCS for details.</td>
</tr>
<tr>
<td>982</td>
<td>CDLF_ERR</td>
<td>Could not resolve the specified function name in the specified DLL or shared library. See CTSTATUS.FCS for details.</td>
</tr>
<tr>
<td>983</td>
<td>ICT_A3T</td>
<td>Client/server mismatch on IICTxtd communications.</td>
</tr>
<tr>
<td>984</td>
<td>ALMT_ERR</td>
<td>Logon is denied because the number of distinct user accounts that are allowed to be connected at one time has been reached.</td>
</tr>
<tr>
<td>985</td>
<td>LDRQ_ERR</td>
<td>Logon is denied because this user account requires LDAP authentication, but c-tree Server has not enabled LDAP authentication.</td>
</tr>
<tr>
<td>986</td>
<td>UNKA_ERR</td>
<td>The specified user account authentication type is invalid or not supported.</td>
</tr>
<tr>
<td>987</td>
<td>LRUB_ERR</td>
<td>LRU scheme for index buffers did not find an available index buffer. Most likely caused by a low IDX_MEMORY.</td>
</tr>
<tr>
<td>-988</td>
<td>CLSF_COD</td>
<td>Internal routine try_FBclsfile successfully closed a low-level index file during an ISAM file block.</td>
</tr>
<tr>
<td>-989</td>
<td>OPNF_COD</td>
<td>Internal routine try_FBopnfile successfully re-opened a low-level index file during an ISAM file block.</td>
</tr>
<tr>
<td>990</td>
<td>DPRT_NSUP_ERR</td>
<td>This operation is not supported for a dynamic partitioned file.</td>
</tr>
<tr>
<td>991</td>
<td>DPRT_NOMBR_ERR</td>
<td>No partition members have been associated with this dynamic partition host file.</td>
</tr>
<tr>
<td>992</td>
<td>ALGN_ERR</td>
<td>Client cannot correct difference between client alignment and the alignment of the data file on disk.</td>
</tr>
<tr>
<td>993</td>
<td>DPRT_DDIF_ERR</td>
<td>The table, index, record, or field definitions of the partition member differ from those of the dynamic partition host file.</td>
</tr>
<tr>
<td>994</td>
<td>SCNV_ERR</td>
<td>Server does not support server-side record image conversion.</td>
</tr>
<tr>
<td>995</td>
<td>VSSI_ERR</td>
<td>Unable to initialize VSS writer support. The 64-bit versions of Windows Vista and later only support native VSS writers (64-bit systems only support a 64-bit VSS writer). See CTSTATUS.FCS for details.</td>
</tr>
<tr>
<td>996</td>
<td>MOFL_ERR</td>
<td>The MULTIOPN_DIFUSR/SAMUSR setting cannot be changed on this file because this connection has other open instances of this file that are using a different MULTIOPN_DIFUSR/SAMUSR setting than the one specified by the caller, or this file has pending locks.</td>
</tr>
<tr>
<td>997</td>
<td>RTGTRM_ERR</td>
<td>A user-defined callback function terminated the RTG operation.</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>998</td>
<td>MUOP_RCR</td>
<td>Attempt to perform multiple file open (co-file) with active recursive locks on the file.</td>
</tr>
<tr>
<td>999</td>
<td>MLOK_ERR</td>
<td>Recursive data file lock not allowed if user has opened the file multiple times (co-files).</td>
</tr>
</tbody>
</table>
### c-treeDB Error and Return Values

This table lists the possible c-treeDB errors that may be encountered while using c-treeDB.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CTDBRET_OK</td>
<td>c-treeDB C API return OK</td>
</tr>
<tr>
<td>4000</td>
<td>CTDBRET_BASE</td>
<td>Base error number</td>
</tr>
<tr>
<td>4001</td>
<td>CTDBRET_NOMEMORY</td>
<td>Not enough memory</td>
</tr>
<tr>
<td>4002</td>
<td>CTDBRET_NULHANDLE</td>
<td>Handle is NULL</td>
</tr>
<tr>
<td>4003</td>
<td>CTDBRET_NOTSESSION</td>
<td>No session handle</td>
</tr>
<tr>
<td>4004</td>
<td>CTDBRET_INVARG</td>
<td>Invalid argument</td>
</tr>
<tr>
<td>4005</td>
<td>CTDBRET_INDEXRANGE</td>
<td>Index out of range</td>
</tr>
<tr>
<td>4006</td>
<td>CTDBRET_ARGSMALL</td>
<td>Argument is too small</td>
</tr>
<tr>
<td>4007</td>
<td>CTDBRET_NULARG</td>
<td>Null argument not valid</td>
</tr>
<tr>
<td>4008</td>
<td>CTDBRET_ARGNUL</td>
<td>Null argument not valid</td>
</tr>
<tr>
<td>4009</td>
<td>CTDBRET_NOTOPEN</td>
<td>Table is not open</td>
</tr>
<tr>
<td>4010</td>
<td>CTDBRET_NOTDATABASE</td>
<td>Not a database handle</td>
</tr>
<tr>
<td>4011</td>
<td>CTDBRET_ISACTIVE</td>
<td>Handle is active</td>
</tr>
<tr>
<td>4012</td>
<td>CTDBRET_NOTACTIVE</td>
<td>Handle is not active</td>
</tr>
<tr>
<td>4013</td>
<td>CTDBRET_NOTTABLE</td>
<td>Handle is not a table</td>
</tr>
<tr>
<td>4014</td>
<td>CTDBRET_NOSUCHFIELD</td>
<td>Unknown field name</td>
</tr>
<tr>
<td>4015</td>
<td>CTDBRET_CANTDELETE</td>
<td>Can’t perform delete</td>
</tr>
<tr>
<td>4016</td>
<td>CTDBRET_FIELDEXIST</td>
<td>Field already exists</td>
</tr>
<tr>
<td>4017</td>
<td>CTDBRET_NOTFIELD</td>
<td>Not a field handle</td>
</tr>
<tr>
<td>4018</td>
<td>CTDBRET_NOTINDEX</td>
<td>Not an index handle</td>
</tr>
<tr>
<td>4019</td>
<td>CTDBRET_INVTYPE</td>
<td>Invalid field/key type</td>
</tr>
<tr>
<td>4020</td>
<td>CTDBRET_NOTSEGMENT</td>
<td>Not a segment handle</td>
</tr>
<tr>
<td>4021</td>
<td>CTDBRET_DATABASEEXIST</td>
<td>Database already exists</td>
</tr>
<tr>
<td>4022</td>
<td>CTDBRET_TABLEEXIST</td>
<td>Table already exists</td>
</tr>
<tr>
<td>4023</td>
<td>CTDBRET_NOSUCHTABLE</td>
<td>Table does not exist</td>
</tr>
<tr>
<td>4024</td>
<td>CTDBRET_NOTRECORD</td>
<td>Not a record handle</td>
</tr>
<tr>
<td>4025</td>
<td>CTDBRET_INTERNAL</td>
<td>Internal error</td>
</tr>
<tr>
<td>4026</td>
<td>CTDBRET_INVFIND</td>
<td>Invalid find mode</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>4027</td>
<td>CTDBRET_NODATA</td>
<td>No data in record</td>
</tr>
<tr>
<td>4028</td>
<td>CTDBRET_INVFORMAT</td>
<td>Invalid date or time format</td>
</tr>
<tr>
<td>4029</td>
<td>CTDBRET_INVDATE</td>
<td>Invalid date</td>
</tr>
<tr>
<td>4030</td>
<td>CTDBRET_INVDAY</td>
<td>Invalid date day</td>
</tr>
<tr>
<td>4031</td>
<td>CTDBRET_INVMONTH</td>
<td>Invalid date month</td>
</tr>
<tr>
<td>4032</td>
<td>CTDBRET_INVYEAR</td>
<td>Invalid date year</td>
</tr>
<tr>
<td>4033</td>
<td>CTDBRET_INVTIME</td>
<td>Invalid packed CTTIME</td>
</tr>
<tr>
<td>4034</td>
<td>CTDBRET_INVHOUR</td>
<td>Invalid hour</td>
</tr>
<tr>
<td>4035</td>
<td>CTDBRET_INVMINUTE</td>
<td>Invalid minute</td>
</tr>
<tr>
<td>4036</td>
<td>CTDBRET_INVSECOND</td>
<td>Invalid second</td>
</tr>
<tr>
<td>4037</td>
<td>CTDBRET_INVAMPM</td>
<td>Invalid morning/evening indicator.</td>
</tr>
<tr>
<td>4038</td>
<td>CTDBRET_OVERFLOW</td>
<td>Operation causes Overflow</td>
</tr>
<tr>
<td>4039</td>
<td>CTDBRET_UNDERFLOW</td>
<td>Operation causes Underflow</td>
</tr>
<tr>
<td>4040</td>
<td>CTDBRET_DIVBYZERO</td>
<td>Division by zero error</td>
</tr>
<tr>
<td>4041</td>
<td>CTDBRET_INVDATETIME</td>
<td>Invalid DateTime</td>
</tr>
<tr>
<td>4042</td>
<td>CTDBRET_CANTCONVERT</td>
<td>Can't perform type conversion</td>
</tr>
<tr>
<td>4043</td>
<td>CTDBRET_TOOBIG</td>
<td>Argument is too big</td>
</tr>
<tr>
<td>4044</td>
<td>CTDBRET_NOSUCHPATH</td>
<td>Path does not exist</td>
</tr>
<tr>
<td>4045</td>
<td>CTDBRET_NOSUCHINDEX</td>
<td>Unknown index number</td>
</tr>
<tr>
<td>4046</td>
<td>CTDBRET_NOTFOUND</td>
<td>Not found</td>
</tr>
<tr>
<td>4047</td>
<td>CTDBRET_INVSEGMODE</td>
<td>Invalid segment mode</td>
</tr>
<tr>
<td>4048</td>
<td>CTDBRET_NOINDEX</td>
<td>Table has no indices</td>
</tr>
<tr>
<td>4049</td>
<td>CTDBRET_NOSUCHSEGMENT</td>
<td>Invalid segment number</td>
</tr>
<tr>
<td>4050</td>
<td>CTDBRET_INVICON</td>
<td>Invalid ISAM context handle</td>
</tr>
<tr>
<td>4051</td>
<td>CTDBRET_INDEXEXIST</td>
<td>Index name already in use</td>
</tr>
<tr>
<td>4052</td>
<td>CTDBRET_MOREDATA</td>
<td>Partial field data read</td>
</tr>
<tr>
<td>4053</td>
<td>CTDBRET_NOINDEXNAME</td>
<td>Invalid index name</td>
</tr>
<tr>
<td>4054</td>
<td>CTDBRET_NOTSUPPORTED</td>
<td>Feature not supported</td>
</tr>
<tr>
<td>4055</td>
<td>CTDBRET_INVLOCKMODE</td>
<td>Invalid lock mode</td>
</tr>
<tr>
<td>4056</td>
<td>CTDBRET_NOLOCK</td>
<td>Record not locked</td>
</tr>
<tr>
<td>4057</td>
<td>CTDBRET_NOWRITELOCK</td>
<td>Record is not locked for writes</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>4058</td>
<td>CTDBRET_NOTDICT</td>
<td>Dictionary handle not allocated</td>
</tr>
<tr>
<td>4059</td>
<td>CTDBRET_NOTYET</td>
<td>Not yet implemented</td>
</tr>
<tr>
<td>4060</td>
<td>CTDBRET_INVNUMBER</td>
<td>Invalid number</td>
</tr>
<tr>
<td>4061</td>
<td>CTDBRET_INVPREC</td>
<td>Invalid number precision</td>
</tr>
<tr>
<td>4062</td>
<td>CTDBRET_INVSSCALE</td>
<td>Invalid number scale</td>
</tr>
<tr>
<td>4063</td>
<td>CTDBRET_INVRECBUF</td>
<td>Record buffer not large enough</td>
</tr>
<tr>
<td>4064</td>
<td>CTDBRET_CANTDELRRECBY</td>
<td>RECBYT index cannot be deleted</td>
</tr>
<tr>
<td>4065</td>
<td>CTDBRET_CANTDELRWID</td>
<td>ROWID index cannot be deleted</td>
</tr>
<tr>
<td>4066</td>
<td>CTDBRET_NOROWID</td>
<td>Table has no ROWID index</td>
</tr>
<tr>
<td>4067</td>
<td>CTDBRET_CANCREATE</td>
<td>Cannot create a file (dupdb)</td>
</tr>
<tr>
<td>4068</td>
<td>CTDBRET_CANTCOPY</td>
<td>Cannot copy a file (dupdb)</td>
</tr>
<tr>
<td>4069</td>
<td>CTDBRET_NOTSUSPENDED</td>
<td>Cannot restore lock</td>
</tr>
<tr>
<td>4070</td>
<td>CTDBRET_INVISOLEVEL</td>
<td>Invalid isolation level</td>
</tr>
<tr>
<td>4071</td>
<td>CTDBRET_CNDXFALSE</td>
<td>Conditional expression evaluates to false</td>
</tr>
<tr>
<td>4072</td>
<td>CTDBRET_CNDXSYNTAX</td>
<td>Conditional expression parser error</td>
</tr>
<tr>
<td>4073</td>
<td>CTDBRET_CNDXTYPE</td>
<td>Type discrepancy</td>
</tr>
<tr>
<td>4074</td>
<td>CTDBRET_CNDXFIELD</td>
<td>Unknown field name</td>
</tr>
<tr>
<td>4075</td>
<td>CTDBRET_CNDXINTERNAL</td>
<td>Internal yacc error</td>
</tr>
<tr>
<td>4076</td>
<td>CTDBRET_CNDXMEMORY</td>
<td>Memory allocation failed</td>
</tr>
<tr>
<td>4077</td>
<td>CTDBRET_CNDXOVERFLOW</td>
<td>Stack overflow</td>
</tr>
<tr>
<td>4078</td>
<td>CTDBRET_CNDXUNDERFLOW</td>
<td>Stack underflow</td>
</tr>
<tr>
<td>4079</td>
<td>CTDBRET_CNDXEXEC</td>
<td>Invalid execution node</td>
</tr>
<tr>
<td>4080</td>
<td>CTDBRET_CNDXDIVISION</td>
<td>Division by zero</td>
</tr>
<tr>
<td>4081</td>
<td>CTDBRET_CNDXNOSCHEMA</td>
<td>No record schema</td>
</tr>
<tr>
<td>4082</td>
<td>CTDBRET_CNDXNORECBUF</td>
<td>No record buffer</td>
</tr>
<tr>
<td>4083</td>
<td>CTDBRET_CNDXSDAT</td>
<td>Not enough data</td>
</tr>
<tr>
<td>4084</td>
<td>CTDBRET_INVSESSIONTYPE</td>
<td>Invalid session type</td>
</tr>
<tr>
<td>4085</td>
<td>CTDBRET_INVALTERATION</td>
<td>Invalid alter table action</td>
</tr>
<tr>
<td>4086</td>
<td>CTDBRET_DIFFERENT</td>
<td>Records are different</td>
</tr>
<tr>
<td>4087</td>
<td>CTDBRET_INVOPERATOR</td>
<td>Invalid operator</td>
</tr>
<tr>
<td>4088</td>
<td>CTDBRET_READONLY</td>
<td>Table was open read only</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Constant</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>4089</td>
<td>CTDBRET_NOTIMPLEMENTED</td>
<td>CALLBACK not implemented</td>
</tr>
<tr>
<td>4090</td>
<td>CTDBRET_INVHANDLE</td>
<td>Invalid handle type</td>
</tr>
<tr>
<td>4091</td>
<td>CTDBRET_INVDCALLBACK</td>
<td>Invalid callback type</td>
</tr>
<tr>
<td>4092</td>
<td>CTDBRET_CANTMOVE</td>
<td>Can't move segment</td>
</tr>
<tr>
<td>4093</td>
<td>CTDBRET_INDEXDUPNAME</td>
<td>Index name already used in database.</td>
</tr>
<tr>
<td>4094</td>
<td>CTDBRET_NOSUCHDATABASE</td>
<td>database does not exit or not found</td>
</tr>
<tr>
<td>4095</td>
<td>CTDBRET_NOSEGMENT</td>
<td>Index has no segments</td>
</tr>
<tr>
<td>4096</td>
<td>CTDBRET_BATCHNOTACTIVE</td>
<td>Batch operation not active</td>
</tr>
<tr>
<td>4097</td>
<td>CTDBRET_BATCHISACTIVE</td>
<td>Batch operation is already active</td>
</tr>
<tr>
<td>4098</td>
<td>CTDBRET_INVBATCHMODE</td>
<td>Invalid batch mode</td>
</tr>
<tr>
<td>4099</td>
<td>CTDBRET_NOTRESOURCE</td>
<td>Not a resource handle</td>
</tr>
<tr>
<td>4100</td>
<td>CTDBRET_INVATTACH</td>
<td>Invalid session attach mode</td>
</tr>
<tr>
<td>4101</td>
<td>CTDBRET_NOTATTACHED</td>
<td>Session not attached</td>
</tr>
<tr>
<td>4102</td>
<td>CTDBRET_BINARYLENGTH</td>
<td>Invalid binary field length</td>
</tr>
<tr>
<td>4103</td>
<td>CTDBRET_ROUTERERROR</td>
<td>Router table error</td>
</tr>
<tr>
<td>4104</td>
<td>CTDBRET_INVROUTERNAME</td>
<td>Invalid router table name</td>
</tr>
<tr>
<td>4105</td>
<td>CTDBRET_INVROUTERNBR</td>
<td>Invalid number of fields in router</td>
</tr>
<tr>
<td>4106</td>
<td>CTDBRET_INVROUTERMAP</td>
<td>Invalid router field mapping</td>
</tr>
<tr>
<td>4107</td>
<td>CTDBRET_DBNOTSQL</td>
<td>The database is not SQL</td>
</tr>
<tr>
<td>4108</td>
<td>CTDBRET_INVBUFFER</td>
<td>The record buffer contains data causing field offset calculations beyond the record</td>
</tr>
</tbody>
</table>

| 4109   | CTDBRET_CALLBACK_1             | User error codes reserved for implementation-specific callback functions. Please refer to documentation that accompanies your specific c-treeDB callback module. |
| 4128   | CTDBRET_CALLBACK_20            |                                                 |

<p>| 4129   | CTDBRET_CANTCHKUID             | Cannot properly check/update dictionary UID      |
| 4130   | CTDBRET_NOMOREVTRES            | No more RESOURCE for VTable available on parent table |
| 4131   | CTDBRET_VTABLEEXIST            | The table has VTable defined in the dictionary    |
| 4132   | CTDBRET_VTABLETYPE             | The VTable type in the dictionary does not match the entry in the resource |
| 4133   | CTDBRET_IDENTITYDEFINED       | An Identity field has been already defined        |</p>
<table>
<thead>
<tr>
<th>Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4134</td>
<td>CTDBRET_NOIDENTITYDEFINED</td>
<td>An Identity field has not been defined</td>
</tr>
<tr>
<td>4135</td>
<td>CTDBRET_FLDISNULL</td>
<td>Field is null</td>
</tr>
<tr>
<td>4136</td>
<td>CTDBRET_RESERVEDNAME</td>
<td>Invalid use of reserved name</td>
</tr>
<tr>
<td>4137</td>
<td>CTDBRET_NOTSUP_IN_DICT</td>
<td>Feature not supported in Dictionary</td>
</tr>
<tr>
<td>4138</td>
<td>CTDBRET_INVFILENAME</td>
<td>Invalid file name.</td>
</tr>
<tr>
<td>4139</td>
<td>CTDBRET_ISVFIELD</td>
<td>Operation not supported on Virtual field.</td>
</tr>
<tr>
<td>4140</td>
<td>CTDBRET_HASVFIELD</td>
<td>Operation not supported on tables with Virtual fields.</td>
</tr>
<tr>
<td>4141</td>
<td>CTDBRET_MISSINGVCLEAR</td>
<td>Cannot add VFields without setting vclear function.</td>
</tr>
<tr>
<td>4142</td>
<td>CTDBRET_MISSINGVGET</td>
<td>Attempting a getfield operation on VField without having a &quot;get&quot; function.</td>
</tr>
<tr>
<td>4143</td>
<td>CTDBRET_MISSINGVSET</td>
<td>Attempting a setfield operation on VField without having a &quot;set&quot; function.</td>
</tr>
<tr>
<td>4144</td>
<td>CTDBRET_INVNUMBLELEN</td>
<td>CTNUMBER has invalid length.</td>
</tr>
<tr>
<td>4145</td>
<td>CTDBRET_INVMILLISECOND</td>
<td>Invalid millisecond</td>
</tr>
<tr>
<td>4146</td>
<td>CTDBRET_INVLEGACYSEGMENT</td>
<td>Unable to compute legacy segment offset (an absolute byte offset cannot be calculated for a legacy segment mode).</td>
</tr>
<tr>
<td>4147</td>
<td>CTDBRET_MSEC_NOTSUPPORTED</td>
<td>Millisecond precision not supported by c-tree data type in use.</td>
</tr>
<tr>
<td>4148</td>
<td>CTDBRET_PARTITION_DEFINED</td>
<td>File partition already defined.</td>
</tr>
<tr>
<td>4149</td>
<td>CTDBRET_NOTFTI</td>
<td>Not a Full Text Index handle.</td>
</tr>
<tr>
<td>4150</td>
<td>CTDBRET_NOSUCHFTI</td>
<td>Unknown Full Text Index number or name.</td>
</tr>
<tr>
<td>4151</td>
<td>CTDBRET_NOFTI</td>
<td>Table has no Full Text Index.</td>
</tr>
<tr>
<td>4152</td>
<td>CTDBRET_FTI_NOFIELD</td>
<td>Full Text Index defined with no associated field.</td>
</tr>
<tr>
<td>4153</td>
<td>CTDBRET_CANTATTACH</td>
<td>Cannot attach (table handle linked to db not supported).</td>
</tr>
<tr>
<td>4154</td>
<td>CTDBRET_NOMOREBATCHES</td>
<td>No room for more batches in the session.</td>
</tr>
</tbody>
</table>
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.

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

### A
- Activating c-treeDB Unicode support ........................................ 209
- Active property ...................................................................... 69
- Add, edit or delete indices ......................................................... 100
- Add, insert, delete, or edit fields .............................................. 99
- Adding an existing Database ..................................................... 71
- Adding an existing table ........................................................... 79
- Adding an existing table under transaction control ..................... 79
- Adding an index to a table ......................................................... 104
- Adding new records .................................................................. 158
- Adding New Resources ............................................................. 177
- Adding or deleting indices ........................................................ 90
- Adding, inserting or deleting fields .......................................... 87
- Additional Resources ............................................................... 12, 29, 41, 59
- Allocating and Freeing Memory Inside Callbacks ....................... 206
- Alter the table ......................................................................... 100
- Altering a table ........................................................................ 99
- Atomicity ............................................................................... 170
- Attach and Detach Existing Sessions ....................................... 75
- Attach and Detach Open Tables ................................................. 106
- Automatic conversion ............................................................. 154
- recovery ............................................................................... 170
- Automatic data type conversion .............................................. 154
- Automatic Recovery ............................................................... 170
- Automatic Type Promotion ..................................................... 137

### B
- Base class ........................................................................... 238
- Batch Modes ................................................................. 145
- Batches with c-treeDB .......................................................... 144
- Bigint
  - class ........................................................................ 282
  - data types ................................................................... 166
- Blob class ........................................................................ 296
- Building a target key ............................................................ 119
- BuildKey ............................................................................ 200

### C
- C Language Equivalents ......................................................... 125
- Callback Function Type ....................................................... 184
- Callback Handle Parameters ............................................... 185
- Callback Return Codes ....................................................... 185
- Callback Types .................................................................... 186
- Changing default properties ................................................ 94
- Check if a record is locked ................................................... 162
- Check if field is fixed or variable length ................................ 157
- Class
  - CTBase ........................................................................ 238
  - CTBigInt .................................................................... 282
  - CTBlob ....................................................................... 296
  - CTCurrency ................................................................... 309
  - CTDATABASE .................................................................. 326
  - CTDigit ......................................................................... 351
  - CTDatetime .................................................................... 369
  - CTException ................................................................. 393
  - CTField ......................................................................... 401
  - CTIndex ......................................................................... 430
  - CTMoney ......................................................................... 462
  - CTNumber ....................................................................... 475
  - CTRecord ......................................................................... 494
  - CTSegment ...................................................................... 631
  - CTSession ....................................................................... 648
  - CTString ......................................................................... 692
  - CTTable ......................................................................... 727
  - CTTIME ......................................................................... 817
- class CTBase ........................................................................ 238
- class CTBigInt ................................................................. 282
- class CTBlob ...................................................................... 296
- class CTCurrency .............................................................. 309
- class CTDatabase .............................................................. 326
- class CTDigit .................................................................... 351
- class CTDatetime .............................................................. 369
- class CTException ............................................................. 393
- class CTField .................................................................... 401
- class CTIndex .................................................................... 430
- class CTMoney ................................................................. 462
- class CTNumber ............................................................... 475
- class CTRecord ................................................................. 494
- class CTResource .............................................................. 609
- class CTSegment ............................................................... 631
- class CTSession ................................................................. 648
- class CTString ................................................................. 692
- class CTTable ................................................................. 727
- class CTTIME .................................................................... 817
- clearing a field .................................................................. 158
- clearing the record buffer ..................................................... 157
- Closing a table .................................................................... 99
- Common functionality ......................................................... 61
- Compatibility ....................................................................... 220
- field mapping .................................................................... 220
- with c-tree Plus ISAM ....................................................... 220
- with c-treeSQL ................................................................. 220
- with low-level data files ....................................................... 220
- Compatibility with c-treeACE ISAM and Low-Level Data Files .................................................. 220
- Compatibility with c-treeACE SQL ........................................ 220
- Compatibility with other c-tree API Technologies ................ 220
- Conditional Expression Callback Function .......................... 142
- Connecting to a database ..................................................... 77
- Constants ........................................................................... 123
- Constructing Resource Objects .......................................... 176
- Converting from Unicode UTF-16 to UTF-8 ......................... 215
- Copyright Notice .................................................................. dccclxxviii
- Creating a Database object ................................................ 77
- Creating a new Database ..................................................... 71
- Creating a new session dictionary ....................................... 66

All Rights Reserved
CTBase Class
Creating a New Table ..............................................87
Creating a record object ...........................................113
Creating a record set ..............................................121
Creating a Session object ...........................................65
Creating a Table object ............................................86
Creating a table object without database support .............86
Creating a table under transaction control ......................96
Creating Key Segments based on Unicode Fields ..................212
Creating tables for transaction processing ......................171
Creating Tables with Unicode Field types .......................210
Creating the table .................................................94

CTBase
CTBigint ........................................................................287
CTBase ........................................................................241
~CTBase ......................................................................242
Abort ...........................................................................243
GetAutoCommit ............................................................244
Begin .........................................................................245
ClearError ....................................................................246
Commit .........................................................................247
GetDefDateType ............................................................248
GetDefFloatFormat .......................................................249
GetDefTimeType ...........................................................250
GetError .......................................................................251
GetHandle .....................................................................252
GetHandleId ..................................................................253
GetKeepLock ..................................................................254
GetLockMode ..................................................................255
GetOperation ..................................................................256
GetSystemConfig ...........................................................257
GetTransactionMode .......................................................261
GetUserTag ....................................................................262
IsLockActive ...................................................................263
IsTransActive ..............................................................264
Lock .............................................................................265
RestoreSavePoint ..........................................................266
SetAutoCommit .............................................................267
SetDefDateType ............................................................268
SetDefFloatFormat .......................................................269
SetDefTimeType ...........................................................270
SetError .......................................................................271
SetHandle ......................................................................272
SetKeepLock ...................................................................273
SetOperation ...................................................................275
SetSavePoint ..................................................................277
SetUserTag ....................................................................278
SetTransactionMode .......................................................279
SwitchInstance .............................................................280
Unlock ..........................................................................281

CTBase Class .................................................................238
CTBase Methods ............................................................239
CTBase Public Members ..................................................238

CTBigint
operator = ....................................................................283
operator <, <=, >, >=, ==, != ........................................284
CTBigint ........................................................................286

~CTBigint ......................................................................287
AsBigInt .......................................................................288
AsFloat .........................................................................289
AsLong .........................................................................290
AsMoney .......................................................................291
AsString .......................................................................292
IsZero ..........................................................................293
SetBigInt ......................................................................294
Zero ..............................................................................295

CTBIGINT ....................................................................166
CTBigInt Class ...............................................................282
CTBigInt Methods ..........................................................285
CTBigInt Operators .........................................................282

CTBlob
operator = ....................................................................297
operator <, <=, >, >=, ==, != ........................................298
CTBlob .........................................................................300
~CTBlob .......................................................................301
AsString .......................................................................302
Get ..............................................................................303
GetBlob .........................................................................304
GetSize .........................................................................305
Resize ............................................................................306
SetBlob .........................................................................307
UpdateBlob .....................................................................308

CTBlob Class .................................................................296
CTBlob Methods .............................................................299
CTBlob Operators ..........................................................296

CTCurrency
operator =, +=, -=, "=\-, /=, abs, +, -, "*, / ........................................310
operator <, <=, >, >=, ==, != ........................................311
CTCurrency ..................................................................313
~CTCurrency ..................................................................314
AsBigInt .......................................................................315
AsCurrency .....................................................................316
AsFloat .........................................................................317
AsLong .........................................................................318
AsMoney .......................................................................319
AsString .......................................................................320
IsZero ..........................................................................321
Round ............................................................................322
SetCurrency ...................................................................323
StringToCurrency ..........................................................324
Zero ..............................................................................325

CTCURRENCY ................................................................167
CTCurrency Class ..........................................................309
CTCurrency Methods .......................................................312
CTCurrency Operators .....................................................309

CTDatabase
CTDatabase ...................................................................328
~CTDatabase ..................................................................329
AddTable .......................................................................330
CloseAll .......................................................................331
Connect .........................................................................332
Create ..........................................................................333
DeleteTable .....................................................................334
Index

CTDate Operators .................................................. 335
CTDate Class .......................................................... 336
FindTable .................................................................. 337
FindTable .................................................................. 338
FirstTable .................................................................. 339
GetFirstActive .......................................................... 340
GetNextActive .......................................................... 341
GetTable .................................................................. 342
GetPath .................................................................. 343
GetTableCount ......................................................... 344
GetTable .................................................................. 345
IsActive .................................................................. 346
IsExclusive ................................................................ 347
NextTable .................................................................. 348
RenameTable ............................................................ 349
SetExclusive ............................................................. 350
CTDatabase Class ...................................................... 326
CTDatabase Methods .................................................. 327
CTDateTime Methods ................................................... 345
CTDateTime Class ........................................................ 346
CTDateTime Operators ................................................ 347
CTDateTime .................................................................. 352
CTException Class ....................................................... 369
CTException Methods .................................................. 372
CTException Operators ................................................. 375
CTException ............................................................. 395
~CTException .......................................................... 396
GetErrCode ................................................................ 397
GetErrMsg ................................................................ 398
GetErrLine ................................................................ 399
GetSourceFile .......................................................... 400
CTException ............................................................. 395
CTException Methods .................................................. 394
CTField .................................................................. 400
operator= ................................................................ 401
ClearFieldDefaultValue ............................................. 404
Second .................................................................. 388
SetDateTime .............................................................. 389
StringToDateTime ...................................................... 390
Unpack .................................................................. 391
Year .................................................................. 392
CTDateTime Class ....................................................... 369
CTDateTime Methods ................................................... 372
CTDateTime Operators ................................................. 375
CTDB_DATABASE_DISCONNECT ......................... 185
CTDB_DATABASE_DISCONNECT ......................... 185
CTDB_ON_SESSION_LOGOUT ........................................ 187
CTDB_ON_DATABASE_CONNECT .................................. 187
CTDB_ON_DATABASE_DISCONNECT ......................... 188
CTDB_ON_RECORD_AFTER_BUILD_KEY186, 200
CTDB_ON_RECORD_AFTER_READ .............................. 186, 198
CTDB_ON_RECORD_AFTER_WRITE ......................... 186, 203
CTDB_ON_RECORD_BEFORE_BUILD_KEY186, 199
CTDB_ON_RECORD_BEFORE_READ .............................. 186, 197
CTDB_ON_RECORD_BEFORE_WRITE ......................... 186, 202
CTDB_ON_RECORD_INIT ............................................. 186, 196
CTDB_ON_RECORD_MAPCHILD .................................. 204
CTDB_ON_RECORD_MAPPARENT .................................. 204
CTDB_ON_RECORD_RESET ........................................... 186, 197
CTDB_ON_SESSION_LOGON ......................................... 185, 186
CTDB_ON_SESSION_LOGOUT ....................................... 185, 187
CTDB_ON_TABLEAlter ................................................ 185, 195
CTDB_ON_TABLE_CLOSE ............................................. 185, 190
CTDB_ON_TABLE_GET_DODA ....................................... 185, 191
CTDB_ON_TABLE_GET_EXT_INFO ............................... 185, 193
CTDB_ON_TABLE_GET_SCHEMA ................. 185, 193
CTDB_ON_TABLE_GET_VTABLE_INFO ....................... 194
CTDB_ON_TABLE_OPEN .............................................. 185, 188
CTDB_ON_TABLE_REBUILD ......................................... 185, 195
cmdbClearAllCallback ............................................. 205
cmdbClearCallback ................................................... 205
cmdbConnect .......................................................... 187
cmdbGetCallback ...................................................... 205
cmdbGetRebuildProgress ......................................... 195
cmdbLogon ............................................................. 186
cmdbOpenTable ........................................................ 188
cmdbSDK.h and c-treeDB return codes ..................... 185
cmdbSetCallback ....................................................... 205
cmdbUNICODE .......................................................... 209
CTException ............................................................. 395
~CTException .......................................................... 396
GetErrCode ................................................................ 397
GetErrMsg ................................................................ 398
GetErrLine ................................................................ 399
GetSourceFile .......................................................... 400
CTException ............................................................. 395
CTException Methods .................................................. 394
CTField .................................................................. 400
operator= ................................................................ 401
ClearFieldDefaultValue ............................................. 404

All Rights Reserved 891 www.faircom.com
CTField Operators .............................................. 405
~CTField ..................................................... 406
GetFieldDefaultValue ......................................... 407
GetFieldDefaultTimeType .................................... 408
GetFieldDefaultValue ........................................ 409
GetLength ..................................................... 410
GetNullFlag .................................................... 411
GetName ....................................................... 411
GetNumber ..................................................... 413
GetPrecision .................................................. 414
GetScale ....................................................... 415
GetStatus ...................................................... 416
GetType ......................................................... 417
IsFieldDefaultValueSet ...................................... 418
IsNumeric ...................................................... 419
SetFieldDefaultDateType .................................... 420
SetFieldDefaultTimeType .................................... 420
SetFieldDefaultValue ........................................ 422
SetLength ...................................................... 424
SetName ....................................................... 425
SetNullFlag .................................................... 426
SetPrecision .................................................. 427
setScale ....................................................... 428
SetType ......................................................... 429
CTField Class .................................................. 401
CTField Methods .............................................. 403
CTField Operators ............................................ 401

CTIndex
operator= ....................................................... 431
CTIndex ....................................................... 433
~CTIndex ...................................................... 434
AddSegment .................................................... 435
DelSegment ..................................................... 436
GetDuplicateFlag ............................................. 437
GetEmptyChar .................................................. 438
GetIndexFileName ............................................ 439
GetIndexKSeg ................................................ 440
GetIdxno ....................................................... 441
GetKeyLength ................................................ 442
GetKeyType .................................................... 443
GetName ....................................................... 444
GetNullFlag .................................................... 445
GetNumber ..................................................... 446
GetSegment .................................................... 447
GetSegmentCount ............................................. 448
GetStatus ..................................................... 449
GetTemporaryFlag ............................................. 450
GetUUID ....................................................... 451
InsertSegment ............................................... 452
MoveSegment .................................................. 453
SetDuplicateFlag ............................................. 454
SetEmptyChar .................................................. 455
SetIndexFileName ............................................. 456
SetIndexKSeg .................................................. 458
SetKeyType .................................................... 459
SetNullFlag .................................................... 460
SetTemporaryFlag ............................................. 461

CTIndex Class ................................................ 430
CTIndex Methods ............................................. 432
CTIndex Operators .......................................... 430
ctKSEG_SSIIZ PROVIDED .................................. 218
crKSEG_SSIIZ_COMPUTED .................................. 218

CTMoney
operator =, +=, -, *, /=, abs, +, -, *, /............... 463
operator <, <=, >, >=, ==, != ............................ 464
CTMoney ...................................................... 466
~CTMoney ..................................................... 467
AsFloat ....................................................... 468
AsLong ......................................................... 469
AsMoney ....................................................... 470
AsString ...................................................... 471
SetInt ........................................................ 472
SetMoney ..................................................... 473
StringToMoney .............................................. 474

CTNUMBER ...................................................... 168
CTNumber Class .............................................. 475
CTNumber Methods .......................................... 478
CTNumber Operators ......................................... 475

CTNumber
operator =, +=, -, *, /=, abs .............................. 476
operator <, <=, >, >=, ==, != ............................ 477
CTNumber ..................................................... 479
~CTNumber ................................................... 480
AsBigInt ...................................................... 481
AsCurrency ................................................... 482
AsFloat ....................................................... 483
AsLong ......................................................... 484
AsMoney ....................................................... 485
AsNumber ..................................................... 486
AsString ...................................................... 487
DecimalDigits ............................................... 488
IntegralDigits .............................................. 489
IsZero ......................................................... 490
Round ........................................................ 491
SetNumber ................................................... 492
Zero .......................................................... 493

CTRecord

CTRecord ..................................................... 498
~CTRecord ................................................... 499
AtPercentile ................................................ 500
BatchLoaded ............................................... 501
BatchLocked ............................................... 502
BatchMode ................................................... 503
BatchTotal .................................................. 504
BuildTargetKey ............................................. 505
Clear ........................................................ 506
ClearField .................................................... 507
Delete ........................................................ 508
Index

EndBatch .............................................. 509
EstimateSpan ........................................ 510
Find .................................................. 512
FindRowid ............................................. 513
FindTarget ............................................ 514
First ................................................. 515
GetDefaultIndex ..................................... 516
GetDefaultIndexName ................................. 517
GetErrorIndex ....................................... 518
GetFieldAddress ..................................... 519
GetFieldAsBigInt .................................... 520
GetFieldAsBlob ...................................... 521
GetFieldAsBool ...................................... 522
GetFieldAsByte ...................................... 523
GetFieldAsChar ..................................... 524
GetFieldAsCurrency .................................. 525
GetFieldAsDate ...................................... 526
GetFieldAsDateTime .................................. 527
GetFieldAsFloat ..................................... 528
GetFieldAsMoney ..................................... 529
GetFieldAsNumber .................................... 530
GetFieldAsShort ..................................... 531
GetFieldAsSigned .................................... 532
GetFieldAsString ..................................... 533
GetFieldAsTime ...................................... 534
GetFieldAsUnsigned ................................. 535
GetFieldAsUTF16 ...................................... 536
GetFieldAsWord ...................................... 537
GetFieldByName ...................................... 538
GetFieldLength ...................................... 539
GetFieldName ........................................ 540
GetFieldOffset ....................................... 541
GetFieldSize ......................................... 542
GetFieldType ......................................... 543
GetFilter ............................................. 544
GetRecordBuffer .................................... 545
GetRecordCount ...................................... 546
GetRecordKeyPos .................................... 547
GetRecordLength ..................................... 548
GetRecordPos ........................................ 549
GetRecordSize ........................................ 550
GetRowid ............................................. 551
InsertBatch ......................................... 552
IsBatchActive ........................................ 554
IsEdited ............................................. 555
IsFiltered ............................................ 556
IsNew .................................................. 557
IsNullField .......................................... 558
IsRangeOn ........................................... 559
IsRecordSetOn ....................................... 560
IsVariableField ...................................... 561
Last .................................................... 562
LockRecord ........................................... 563
Next .................................................... 564
NextBatch ............................................ 565
NextInBatch .......................................... 566
NumberOfKeyEntries ................................ 567
Prev ................................................... 568
RangeOff ............................................. 569
RangeOn .............................................. 570
Read .................................................... 571
RecordSetOff ........................................ 572
RecordSetOn ......................................... 573
Reset ................................................... 574
SeekRecord ........................................... 575
SetBatch .............................................. 576
SetDefaultIndex ..................................... 581
SetEdited ............................................ 582
SetFieldAsBigInt .................................... 583
SetFieldAsBlob ...................................... 584
SetFieldAsBool ...................................... 585
SetFieldAsByte ...................................... 586
SetFieldAsChar ...................................... 587
SetFieldAsCurrency .................................. 588
SetFieldAsDate ....................................... 589
SetFieldAsDateTime .................................. 590
SetFieldAsFloat ..................................... 591
SetFieldAsMoney ..................................... 592
SetFieldAsNumber .................................... 593
SetFieldAsShort ..................................... 594
SetFieldAsSigned ..................................... 595
SetFieldAsString .................................... 596
SetFieldAsTime ....................................... 597
SetFieldAsUnsigned .................................. 598
SetFieldAsUTF16 ..................................... 599
SetFieldAsWord ...................................... 600
SetFilter ............................................. 601
SetNew .............................................. 603
SetRecordOffset ..................................... 604
SetRecordPos ........................................ 605
SwitchContext ....................................... 606
UnlockRecord ........................................ 607
Write .................................................. 608
AtPercentile ......................................... 500
EstimateSpan ........................................ 510
RangeOff ............................................. 569
RangeOn .............................................. 570
SwitchContext ....................................... 606
CTRecord Class ....................................... 494
CTRecord Methods .................................... 495
c-treeACE Error Code Reference .................... 847
c-treeACE Expression Parser and Grammar ...... 123
c-treeACE Unicode UTF-16 Field Types .......... 210
c-treeDB callbacks
CTDB_DATABASE_CONNECT ......................... 185
CTDB_DATABASE_DISCONNECT ...................... 185
CTDB_ON_DATABASE_CONNECT ..................... 187
CTDB_ON_DATABASE_DISCONNECT ................. 188
CTDB_ON_RECORD_AFTER_BUILD_KEY ............. 186, 200
CTDB_ON_RECORD_AFTER_READ 186, 198
CTResource Methods

- CTResource ............................................. 611
- Add .................................................. 612
- Delete .................................................. 614
- Find .................................................. 615
- First ................................................... 617
- GetData ................................................ 618
- GetDataLength ....................................... 619
- GetName ............................................... 620
- GetNumber .......................................... 621
- GetType .............................................. 622
- IsLocked ............................................. 623
- Next ................................................ 624
- SetData ............................................. 625
- SetName ............................................. 626
- SetNumber .......................................... 627
- SetType ............................................. 628
- Unlock ............................................... 629
- Update .............................................. 630

CTResource Class ........................................... 609

CTResource Methods ........................................ 610

CTDB_ON_RECORD_AFTER_WRITE 186, 203
CTDB_ON_RECORD_BEFORE_BUILD_K EY .............................................. 186, 199
CTDB_ON_RECORD_BEFORE_READ 186, 197
CTDB_ON_RECORD_BEFORE_WRITE 186, 202
CTDB_ON_RECORD_INIT ................. 186, 196
CTDB_ON_RECORD_RESET ............... 186, 197
CTDB_ON_SESSION_LOGON ............. 185, 186
CTDB_ON_SESSION_LOGOUT .......... 185, 187
CTDB_ON_TABLE_ALTER ............... 185, 189
CTDB_ON_TABLE_CLOSE ............... 185, 190
CTDB_ON_TABLE_GET_DODA .......... 185, 191
CTDB_ON_TABLE_GET_EXT_INFO ...... 185, 193
CTDB_ON_TABLE_GET_RECLLEN ...... 185, 193
CTDB_ON_TABLE_GET_SCHEMA ...... 185, 191
CTDB_ON_TABLE_OPEN ............... 185, 188
CTDB_ON_TABLE_REBUILD .......... 185, 195
memory allocation concerns .......... 206

CTSEG_UNCSEC ........................................... 212
functions
- ctdbSetFieldAsUTF16 .................. 211
- SetTableKSeg ..................................... 217

Unicode
- ctdbUNICODE support .................... 209
Unicode support .................................. 207
Writing UTF-16 Field Data ............ 211
c-treeDB C++ API - Working with Resources .... 176
c-treeDB C++ API Class Reference .... 226
c-treeDB C++ API Example ............. 210
c-treeDB C++ API Methods ............. 217
c-treeDB C++ API UTF-8 Compliance .... 208
c-treeDB Definitions ...................... 227
c-treeDB Error and Return Values ..... 881
c-treeDB Support for Unicode .......... 207
c-treeDB, ISAM, and Low-Level Integration .... 222

CTSession
- CTSession ............................................ 650
- ~CTSession ........................................ 651
- AddDatabase ...................................... 652
- Attach ............................................. 653
- BeginImpersonation ...................... 654
- Create ............................................. 655
- CreateDatabase .................................. 656
- DeleteDatabase .................................. 657
- Detach ............................................. 658
- DisconnectAll ................................... 659
- DropDatabase .................................... 660
- EndImpersonation ......................... 661
- FindActive ....................................... 662
- FindDatabase .................................... 663
- FirstDatabase .................................... 664
- GetAttachMode ................................... 665
- GetDatabaseCount ......................... 666
- GetDatabaseUID ................................. 667
- GetFirstActive ................................. 668
- GetLibType ....................................... 669
- GetLogonOnly .................................... 670
- GetNextActive ................................. 671
- GetParam ......................................... 672
- GetPassword .................................... 673
- GetPath .......................................... 674
- GetPathPrefix ................................. 675
- GetServerName ................................. 676
- GetUserLogonName ....................... 677
- IsActive ......................................... 678
- IsExclusive ..................................... 679
- Logon ............................................. 680
- Logout ........................................... 681
- NextDatabase .................................... 682
- SetConfigurationFile .................... 683
- SetCurrentTextNode ....................... 684
- SetKSegDefaults ......................... 645
- SetMode ........................................... 646
- SetSegmentKSeg ............................... 647
- CTSegment Class ............................... 631
- CTSegment Methods ....................... 633
- CTSegment Operators ..................... 631
- CTSEG_UNCSEG C++ API Example .......... 213
Index

Rebuild ................................................. 795
Remove ................................................. 797
ResetAll .................................................. 798
SetDataDefaultExtentSize ............................ 799
SetDataExtension ........................................ 800
SetGroupid ............................................. 801
SetIndexDefaultExtentSize ......................... 802
SetIndexExtension ...................................... 803
SetIndexFileName ...................................... 804
SetPadChar ............................................. 805
SetPassword ........................................... 806
SetPath .................................................. 807
SetPermission .......................................... 808
SetTableKSeg .......................................... 809
SetOwner ............................................... 810
SystemFilterOff ....................................... 811
SystemFilterOn ........................................ 812
UpdateCreateMode ..................................... 814
UpdatePadChar ......................................... 815
UnlockTable ............................................ 816
Attach .................................................... 821
AttachXtd ............................................... 822
ClearAllFieldDefaultValue ......................... 824
Detach ................................................... 828
GetDatno ............................................... 829
GetIdxno ............................................... 830
GetTableKSeg ......................................... 831
MoveSegment .......................................... 835
Remove ............................................... 837
SetTableKSeg .......................................... 839
SystemFilterOff ....................................... 842
SystemFilterOn ........................................ 843
Remove .................................................. 797
CTTable Class ........................................... 727
CTTable Methods ....................................... 728
CTTime
operator = ............................................. 818
operator <, <=, >, >=, ==, != ....................... 819
CTTime .................................................. 825
~CTTime ............................................... 826
CurrentTime .......................................... 827
GetTime .................................................. 832
Hour ...................................................... 833
Minute ................................................... 834
Pack ....................................................... 836
Second ................................................... 838
SetTime .................................................. 840
StringToTime .......................................... 841
TimeToString .......................................... 845
Unpack .................................................... 846
CTTime Class ........................................... 817
CTTime Methods ....................................... 820
CTTime Operators ..................................... 817
ciUNICODE .............................................. 209
Currency
class ..................................................... 309
data types ............................................... 167
Custom Application Expressions ................. 138
D
Data
conversion .............................................. 154
file compatibility ..................................... 220
Data file extent size .................................. 108
Data integrity .......................................... 170
atomicity ................................................. 170
automatic recovery .................................... 170
create table for transaction processing .......... 171
locking .................................................... 172
save points ............................................. 172
start transaction ...................................... 171
terminate transaction .................................. 171
transactions ............................................ 170
Data Integrity .......................................... 170
Data types .............................................. 163
bigint .................................................... 166
currency ................................................ 167
date ....................................................... 163
date/time ............................................... 165
money .................................................... 166
number ................................................... 168
numeric ................................................. 166
scalar ..................................................... 163
time ........................................................ 165
timestamp ............................................... 165
Data Types .............................................. 163
Database Dictionary .................................. 84
Database name ........................................ 78
Database path ........................................... 78
Database properties .................................. 78
Database UID (Unique IDentifier) ............... 74
Databases .................................................. 60, 185
add existing .......................................... 71
add table under transaction control ............. 79
add tables .............................................. 79
class ..................................................... 326
connect .................................................. 77
create new ............................................. 71
create object ......................................... 77
delete .................................................... 72
delete table ............................................ 80
delete table under transaction control .......... 81
dictionary .............................................. 84
drop ...................................................... 71
drop table .............................................. 80
drop table under transaction control .......... 80
find ....................................................... 72
find active ............................................. 73
find active by UID ..................................... 74
find active table ..................................... 82
find active table by UID ......................... 83
find by UID ............................................. 74
ICU International Components for Unicode

Index

Finding records by ROWID ............................................. 120
Finding records by target key ........................................ 120
First Active Database .................................................. 73
First Active Table ......................................................... 82
First Database ............................................................. 72
First record ................................................................. 116
First Table ................................................................. 81
Fixed or variable length records .................................... 88
Float formats ................................................................ 63
Forcing a table rebuild .................................................. 106
Forcing an index rebuild ................................................ 105
Formats
  default date ................................................................ 63
  float ............................................................................. 63
  time ............................................................................. 63
Freeing locks .................................................................. 174
Freeing locks associated with a table .............................. 174
functions
  c-tree Plus
    BuildKey .................................................................. 200
c-treeDB
  ctdbClearAllCallback .................................................. 205
  ctdbClearCallback ....................................................... 205
  ctdbConnect ............................................................... 187
  ctdbGetCallback ........................................................ 205
  ctdbGetRebuildProgress .............................................. 195
  ctdbLogon ................................................................. 186
  ctdbOpenTable ............................................................ 188
  ctdbSetCallback ........................................................ 205
  ctdbSetFieldAsUTF16 .................................................. 211
  SetTableKSeg ............................................................. 217
Functions
  common ...................................................................... 61
  default date formats .................................................... 63
  error handling ............................................................. 61
  float formats ............................................................... 63
  session wide locking ................................................... 62
  time formats .............................................................. 63
  transaction processing ............................................... 61
  user defined tags ........................................................ 64

G
Get and Set Resource Properties with the
  c-treeDB C++ API ......................................................... 182
GetIndexKSeg ............................................................... 217
GetSegmentKSeg .......................................................... 217
GetTableKSeg ............................................................... 217
Group ID
  table .......................................................................... 108

H
Hidden fields ............................................................... 89
How to Specify a Unicode Key Segment ......................... 212

I
ICU - International Components for Unicode .................. 209
ICU Collation Option Overview ..................................... 213

K
Key
  target ....................................................................... 119
  types ......................................................................... 231

L
Last record ................................................................. 120
Layout of this Manual .................................................. 2
Lock modes ................................................................. 173
Locking ....................................................................... 172
Locks
  check if record is locked .......................................... 162
  demote record locks ............................................... 162
  free .......................................................................... 174
  free locks associated with table ................................ 174
  mode ......................................................................... 173
  record ....................................................................... 161
  record modes .......................................................... 231
  session wide ........................................................... 62
  session wide modes .................................................. 231
  start ......................................................................... 172

M
Manage ............................................................. 8, 22, 36, 51
Managing Databases .................................................. 70
Managing Tables ......................................................... 79
Mathematical Functions ............................................... 133
Money
  class ..................................................................... 462
  data types ............................................................... 166
mtnmake
  Unicode option ........................................................ 209
**Index**

- write field data to buffer ........................................... 153
- Recovery
  - automatic ................................................................. 170
- Relationships .................................................................. 13
- Resetting a record ....................................................... 114
- Resource Identification ............................................... 175
- Resource Locks ............................................................ 182
- Resource Name ............................................................. 176
- Resource Number .......................................................... 176
- Resource Type .................................................................. 175
- Retrieving batch properties ........................................... 152
- Retrieving records .......................................................... 151
- Retrieving records by index range ................................... 147
- Retrieving records by partial key .................................... 146
- Retrieving records by physical order ................................ 149
- Return values
  - c-treeDB ....................................................................... 881
- RFC 2871 ......................................................................... 207
- RFC 3629 ......................................................................... 208
- ROWID index .................................................................... 93

**S**

- Save points ...................................................................... 172
- Save Points ....................................................................... 172
- Scalar data types ............................................................ 163
- Scalar Types ...................................................................... 163
- Security
  - group ............................................................................ 109, 236
  - owner ............................................................................ 109, 236
  - world .............................................................................. 109, 236
- Seek to record .................................................................... 118
- Segment Modes .................................................................. 92, 232
- Segments
  - class .............................................................................. 60
  - modes ............................................................................. 631
  - modes ........................................................................... 92
- Selecting the RECBYT index ............................................ 115
- Selecting the ROWID index ............................................. 116
- Server, user name and password properties ....................... 69
- Session Dictionary .......................................................... 76
- Session logon and logout ................................................ 67
- Session properties ........................................................... 68
- Session Wide Lock Modes ............................................... 231
- Session Wide Locking ..................................................... 62
- Sessions
  - active property ............................................................ 60, 185
  - add database .................................................................. 71
  - class ............................................................................... 648
  - create new database .................................................... 71
  - create new dictionary .................................................. 66
  - create object ................................................................... 65
  - database UID .................................................................. 74
  - delete database ............................................................ 72
  - dictionary file layout .................................................... 76
  - drop database ............................................................... 71
  - find active database ..................................................... 73
  - find active database by UID .......................................... 74
- find database ..................................................................... 72
- find database by UID ....................................................... 74
- first active database .......................................................... 73
- first database .................................................................... 72
- logon .................................................................................. 67
- logout ................................................................................ 67
- managing databases ....................................................... 70
- next active database .......................................................... 73
- next database ..................................................................... 72
- password properties .......................................................... 69
- path property .................................................................... 70
- properties .......................................................................... 68
- server properties .............................................................. 69
- user name properties .......................................................... 69
- without dictionary support ............................................... 74
- working with ..................................................................... 65
- SetIndexKSeg .................................................................... 217
- SetKSegDefaults .............................................................. 217
- SetSegmentKSeg ............................................................... 217
- Sharing the same context .................................................. 114
- Specifying a Unicode Key Segment C++ Example .............. 213
- Specifying a Unicode Key Segment with
  - CTSEG_UNCSEG .......................................................... 213
- Starting a new batch operation ......................................... 146
- Starting a transaction ...................................................... 171
- Starting locks .................................................................... 172
- Storing UTF-8 Data .......................................................... 214
- String and Text Functions ................................................ 126
- String class ....................................................................... 692
- Switching c-tree instances ................................................. 222
- Switching ISAM contexts .................................................. 223

**T**

- Table count ....................................................................... 79
- Table Create Modes .......................................................... 234
- Table file extension ........................................................... 108
- Table group ID ................................................................... 108
- Table name ......................................................................... 107
- Table Open Modes ............................................................ 235
- Table password ................................................................. 108
- Table Path ......................................................................... 107
- Table permission mask ..................................................... 109
- Table Permissions ............................................................. 236
- Table Properties ................................................................. 107
- Table UID (Unique IDentifier) ............................................ 83
- Tables ............................................................................. 60, 185
  - add .................................................................................. 79
  - add fields ......................................................................... 87, 99
  - add index ......................................................................... 90, 100, 104
  - add under transaction control ......................................... 79
  - alter ................................................................................ 99, 100
  - change properties .......................................................... 94
  - class ............................................................................... 727
  - close .............................................................................. 99
  - count tables in database ................................................. 79
  - create ............................................................................. 94
Variables .......................................................... 124

W
Working with Callbacks ........................................ 184, 205
Working with Databases ......................................... 77
Working with Records .......................................... 113
Working with Resources ....................................... 174
Working with Sessions ......................................... 65
Working with Sessions without Dictionary Support .......................................................... 74
Working with Tables ........................................... 85
Working with Unicode .......................................... 207
Writing UTF-16 Data C++ Example ....................... 212
Writing UTF-16 Field Data .................................... 211