



Faircom[®]

c-tree **Plus**[®]
V9

c-tree Driver
Developer's Guide

c-tree Driver

Developer's Guide



FairCom[®]

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

Information in this document is subject to change without notice.

Trademarks

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

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

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

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

6/26/2008

CONTENTS

c-tree Drivers	1
1.1 Items for Consideration.....	1
1.2 Driver Distribution	2
VENDOR.DB.....	2
Notes from your Software Provider.....	2
c-tree Driver SDK.....	2
1.3 Data Alignment/Padding	3
1.4 Supported Data Types	4
1.5 Field Options	4
1.6 Debugging Options	5
1.7 OTP Files	5
OTP File Requirements	5
OTP File Organization	6
OTP File Layout	6
OTP File Contents	6
Data File Description Record.....	6
Field Description Record.....	7
Index File Description Record.....	8
Optional Index Member Record.....	9
Key Segment Description Record.....	10
ctree ODBC Driver c-tree Plus Edition	11
2.1 Supported Data Types	11
2.2 Suppression of ctree dialog boxes.....	13
ctree® Crystal Reports™ Driver	15
3.1 Supported Data Types	15
ctree ODBC Driver - c-tree V4 Edition	17
4.1 Requirements.....	17
4.2 Supported Data Types	17
4.3 c-tree V4 Driver with LONGVARD support off	19
ctree Driver SDK	21
5.1 Installation	21
5.2 Example 1: Compiling and Checking Your Custom DLL	22
5.3 Source Code Overview	22
Primary data I/O Functions (read/write or get/set).....	23
Type Definition Functions	23
Type Verification Functions.....	23
Optional (Sample) Functions	23
5.4 Function Descriptions	23
OT_SETIT	23
OT_GETIT.....	24
OT_STMSP.....	24
OTCtreeTypeToSqlType.....	24
OTSqLTypeToCtreeType.....	25
OT_GETIFIL.....	25
OT_GETDODA	25
OT_SETDK.....	26
OT_GETDK.....	26

Table of Contents

OT_STFLEN	27
OT_CHKTYP.....	27
OT_FLDLEN	27
OT_AlignAdjust	27
OT_Connect.....	28
OT_Disconnect	28
5.5 Example 2: Manipulating Data Read from a File	28
5.6 Example 3: Altering Data	29
5.7 Debugging.....	30

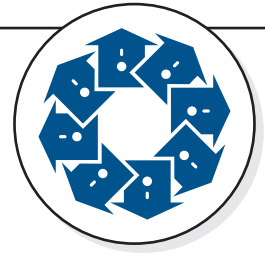
Index.....	31
-------------------	-----------

FAIRCOM TYPOGRAPHICAL CONVENTIONS

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

The following formatted items identify special information.

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



c-tree Drivers

The c-tree® ODBC Driver - c-tree Plus® Edition, the c-tree ODBC Driver - c-tree® V4 Edition, and the c-tree Crystal Reports™ Driver provide access to your c-tree Plus or c-tree data from third-party applications, such as Seagate Crystal Reports™, Microsoft® Access, or your own ODBC-compliant application.

This guide assumes an intermediate level of knowledge of FairCom database engine products, is intended for c-tree and c-tree Plus developers only, and is not intended as an end-user document. Before reading this guide we recommend reading the appropriate FairCom end-user Driver Guide(s) and working through the tutorial(s). Select the correct driver for your need from the table below:

	ODBC-Compliant applications (including Crystal Reports)	Crystal Reports only
c-tree Plus Standard and Extended files	c-tree ODBC Driver - c-tree Plus Edition	c-tree Crystal Reports™ Driver
c-tree V4 files	c-tree ODBC Driver - c-tree V4 Edition	

This chapter describes developer considerations for all the c-tree Drivers, including c-tree Plus/c-tree file requirements, distribution considerations, data type standards, and debugging tips. The chapters that follow describe considerations specific to each Driver. The final chapter provides documentation of the c-tree Driver SDK (sold separately) for informational purposes only.

1.1 Items for Consideration

The c-tree Drivers require the following:

1. c-tree Plus Standard or Extended files or c-tree V4.1F-V4.3 files, depending on the product.
2. The file definitions for the file(s) to be accessed must be stored in either:
 - a) Resource records using Incremental ISAM file definitions (*IFIL*, *IIDX*, *ISEG* arrays). This is the default for c-tree Plus. Unless explicitly disabled, c-tree Plus will add the Incremental ISAM resources automatically. If the resources are not present, use **EnableCtResource()** and **PutIFIL()** to add the Incremental ISAM file definitions.
 - b) o-tree Parameter (OTP) files, similar to ISAM parameter files. See details in [“OTP Files”](#) below. The c-tree ODBC Driver - c-tree V4 Edition requires OTP files.
3. The field definitions, also referred to as a record schema or DODA (dynamic object definition array), must be stored in either
 - a) A *DODA* resource added to each data file with **PutDODA()**.
 - b) o-tree Parameter files.

Note: Mixing OTP files with resources in a database consisting of many data files will work, provided each data file uses either resources or OTP files. Each individual data file and its associated indices must be defined either by resources (*IFIL*, *IIDX*, *ISEG*, and *DODA*) or by an OTP file. Using one method or the other in all data files in the database is the simplest solution.

1.2 Driver Distribution

FairCom encourages application developers to bundle the c-tree Drivers with their applications. This eliminates the need for end-users to create their own dictionary and makes the c-tree Drivers much easier to use. Consider the following points when packaging a c-tree Driver with an end-user application:

The c-tree Driver guides instruct the end-user to check the c-tree Driver disk for an existing *VENDOR.DB* file and refers the reader to the inside cover of the c-tree Driver Guide for a document titled "Notes from your Software Provider". This provides you, the developer, an opportunity to predefine the FairCom Data Dictionary script. By predefining the dictionary script and providing additional instructions, you minimize the possibility of difficulty during the installation process.

VENDOR.DB

If your application stores the c-tree Plus data and index files in a predefined directory structure, generate an import script for your end-users. After creating the import script, rename it to *VENDOR.DB* and copy the file to the root directory of the c-tree Driver disk. Since your application can be installed on different logical drives (i.e., C:, D:, . . .), you may need to include instructions for the end-user to modify the logical drive specifiers in the import script.

Details on building the dictionary import scripts are provided in "Script Method" of the *c-tree ODBC Driver* (both the c-tree Plus and c-tree V4 Edition) and the *c-tree Crystal Reports Driver Guide*, or in "Data Dictionary Creation - Import Method" of *c-tree ODBC Driver Guide* ("Data Dictionary Creation - Import Method" in case of *c-tree Crystal Reports Driver Guide*).

Notes from your Software Provider

"Notes from your Software Provider" is a vendor-created document giving the end-user information specific to a vendor application. Items to consider for this document include:

- Whether you have provided an application-specific import script (*VENDOR.DB*).
- To facilitate the building of complex multi-file queries, we recommend you include your application's data and index file relationship information. The tutorials in "Quick Start" of *c-tree ODBC Driver* (both the c-tree Plus and c-tree V4 Edition) and *c-tree Crystal Reports Driver Guide* show an example layout for providing this information.
- Contact information for your technical support department. To assure consistent information and communication with your customers, FairCom's policy is to provide support to you and to refer your customers to you for support services.
- Any other relevant information that may assist the user in operating the c-tree Driver with your application. Suggestions appear throughout this chapter.

c-tree Driver SDK

The c-tree Driver SDK, purchased separately, includes the c-tree Driver data type conversion routines in full C source code. This allows the application developer to support data types not inherently supported by the c-tree Drivers.

The c-tree Driver SDK gives you the ability to:

- Define special type mappings between c-tree Plus types and c-tree Driver types.
- Define data type capacities.
- Define your own appropriate field level padding.
- Intercept each column's data as it is transferred between c-tree Plus and the c-tree Driver.
- Intercept c-tree Plus records just before they are written to disk.
- Intercept c-tree Plus records just after they are read from disk.
- Handle special conversions between your data stored on disk and the c-tree Driver.
- Define a column's maximum capacity, scale and precision.

An additional benefit of the c-tree Driver SDK is the ability to create and duplicate your own customized Drivers. This will allow you to tailor the installation for your customer's needs and modify the look and feel of the Driver to match your product line. This added flexibility requires an OEM distribution agreement. Contact FairCom for c-tree Driver SDK licensing and pricing information. This product is documented in ["c-tree Driver SDK"](#) for informational purposes only.

1.3 Data Alignment/Padding

The c-tree Driver Setup dialogs have Alignment and Padding buttons for specifying the alignment and padding of the c-tree Plus files.

The recommended alignment option is Default Alignment. This instructs the c-tree Drivers to read the alignment from the c-tree Plus file resource or OTP entry. If for some reason, the alignment has changed since the file was created (i.e., the file was moved to a new platform) the Alignment option provides a mechanism for overriding the default value.

The recommended padding option is Default Padding. This instructs the c-tree Drivers to read the padding from the c-tree Plus file resource or OTP entry. If for some reason, the padding has changed since the file was created (i.e., the file was moved to a new platform) the padding options provide a mechanism for overriding the default value.

There are two padding options for strings: data padding and key padding. Data padding options are "NULLs", "Spaces", or "Spaces w/NULL Trm". This option determines how string data is padded, and defaults to NULLs. String key padding options are "Default", "NULLs", or "Spaces". This option determines how key segments formed over string fields are padded if the underlying string data is shorter than the key segment. "Default" indicates that the pad value in the data file's schema map will be used.

Note: The logic used to determine whether or not to pad key segments is the same as used by the c-tree Plus function `frmkey()`, so that target keys will be properly formed. The c-tree Drivers pad keys formed over string segments only if one or more of the following conditions holds:

- The key segment mode is *SCHSEG* or *USCHSEG* and the underlying field is any string type EXCEPT *CT_FSTRING*.
OR
- The key segment mode is *VSCHSEG* or *UVSCHSEG*.

Otherwise, the key segment is formed directly from the contents of the string field. There is currently no way to configure the c-tree Driver to use different types of padding for different string fields. The c-tree Driver pads every string field in the data file the same way--using the data and key padding options in the c-tree Driver Setup dialog. If a developer must use multiple padding methods in a single data file, the c-tree Driver SDK provides this flexibility.

The key padding option only affects how the target keys are padded. The c-tree Plus ISAM functions to add and update records always use the padding character from the schema map.

1.4 Supported Data Types

Additional details appear in the Driver-specific sections, but the following applies to all Drivers.

Float Types

The float types recognize both decimal and scientific notation; however, mixed notations in one column are not supported.

String Lengths

The maximum allowed length for string fields has not yet been determined. Here are the theoretical (ideal) maximums:

- *CT_STRING CT_FSTRING* - limited only by memory/file system
- *CT_FPSTRING, CT_PSTRING* - 254 characters maximum
- *CT_F2STRING, CT_2STRING* - 65,533 characters maximum
- *CT_F4STRING, CT_4STRING* - 4,294,967,291 characters maximum

1.5 Field Options

The c-tree Drivers support field options, including required, hidden, and read-only fields, as well as precision and scale. Each option is implemented with special information in the field name. However, a field name may not include a period '.'. Only the Hidden Fields option applies to read-only Drivers.

Required Fields

The c-tree Drivers will not allow a NULL value for a field whose name in the *DODA* begins with an asterisk, '*', or a field created using the NOT NULL column constraint. Override this feature with the c-tree Driver SDK.

Hidden Fields

The c-tree Drivers will not display a field whose field name given in the *DODA* begins with an underscore, '_'. Hiding fields that are used to form keys is not recommended and may cause problems.

Read-Only Fields

The c-tree Drivers will display but will not allow updates on a field whose field name given in the *DODA* begins with a right square bracket character, ']'.

Read-Only On Update

To allow a field's value to be set on an insert operation but not on an update, set the field attribute *otOTREE_RDONLYONUPDATE* by OR-ing this value into the field's *frsv* value in the *DODA*. When set, the c-tree ODBC Drivers enforce read-only access on update operations for the field but allows the field value to be set on insert operations. An SQL command that attempts to update such a field will fail with the error "access denied". The *otOTREE_RDONLYONUPDATE* define is included in the c-tree Driver SDK in *otcustm.h*. The c-tree Driver SDK is required to use this feature.

Precision and Scale

The c-tree Drivers store precision and scale setting specified when creating numeric columns. The information is stored by appending it to the specified field name in the format *fieldname, prec, scale* where *fieldname* is the specified field name, and *prec* and *scale* are the specified precision and scale, respectively.

1.6 Debugging Options

The c-tree Driver Setup dialogs have Debug buttons for advanced debugging information if problems are encountered.

When “c-tree Plus Debug” is enabled, the log file contains information about the layout of the open files (i.e., *DODA*) and may contain information about any encountered errors.

The Debug Indexes option adds information specific to debugging index-related problems such as expected data not being retrieved.

To enable either debug option, do the following:

1. Open the c-tree Driver Setup dialog as described in the appropriate Driver Guide.
2. Once in c-tree Setup, click the **Options >>** button.
3. Enable the desired debug box.
4. Click **OK** to close the dialog.

When an application loads the c-tree Driver, the debugging will be enabled as described above.

The debug information will be placed in a log file in the Data Dictionary Path. The log file will be *ctodbc.log* for the c-tree ODBC Driver or *ctcrw.log* for the c-tree Crystal Reports Driver.

Note: The log file can grow very large quickly. For this reason, FairCom recommends using the Debug option only when necessary. This limits the size of the log making it easier to find relevant log entries and saving drive space.

1.7 OTP Files

OTP files allow access to c-tree or c-tree Plus data files without embedded resources. This feature is required for the c-tree ODBC Driver - c-tree V4 Edition, however, this feature is available for all versions of the c-tree Drivers.

OTP File Requirements

To implement the c-tree Drivers without *IFIL* and *DODA* resources, you must:

- Create an o-tree Parameter, OTP, file for each of your application's data files.
- Create a *VENDOR.DB* file listing the OTP files that define the structure of your application's data and index files.
- Create a “Notes from your Software Provider” document explaining any altered installation procedures.

OTP File Organization

OTP files are ASCII text files specifying the characteristics of the data files and indices used in an application program. An OTP file is required for each data file that is not using resources. While these files are similar to ISAM parameter files used by c-tree Plus, OTP and parameter files are not interchangeable.

An OTP file consists of five types of records:

- Data File Description Record
- Field Description Records
- Index File Description Records
- Optional Index Member Records
- Key Segment Description Records

Each OTP file begins with one Data File Description Record specifying the number of fields and indices it uses. This is followed by a Field Description Record for each field in the record. For each index there is a group of records beginning with the Index File Description Record or Index Member Record followed by one or more Key Segment Description Records.

The overall organization of the records is shown in the following schematic and reflects the hierarchical relationship among the data files and indices:

OTP File Layout

```
Data File Description Record
  First Field Description Record
  :::
  Last Field Record
  First Index File Description Record
    First Key Segment Description Record
    :::
    Last Key Segment Record
  Optional Index Member Records
    First Key Segment Description Record
    :::
    Last Key Segment Record
  :::
  Last Index File or Index Member Record
    First Key Segment Description Record
    :::
    Last Key Segment Record
```

OTP File Contents

Each field comprising the OTP file records is separated from neighboring fields by one or more “white space” characters: blanks, tabs, or new lines. In the following subsections, if a parameter corresponds to a formal parameter in a c-tree Plus function call, the parameter name is included after the parameter’s brief description (e.g., *filnam* in the first position of the data file description record).

Data File Description Record

Each Data File Description Record consists of eight required fields, the first four fields corresponding to parameters of **CreateDataFile()**:

- data file name (*filnam*)
- record length (*datlen*)
- file size extension parameter (*xtdsiz*)

- file mode (*filmod*)
- number of associated index files (*num_idx*)
- number of fields (*#flds*)
- symbolic name (without spaces) of the first field in the data record (*fst fldnam*)
- symbolic name of the last field in the data record (*lst fldnam*)

filnam	datlen	xtdsiz	filmod	num_idx	#flds	fst fldnam	lst fldnam
custordr.dat	24	4096	1	2	5	_o_delflag	co_custnumb

Parameters

- *filnam*
The name of the data file, including its extension and (optional) path. For the c-tree V4.x Driver, *filnam* cannot include a path.
If a path is not specified, the path for the OTP file as defined in the data dictionary script (*VENDOR.DB*) file will be used for the path of the data and index files. The data file name must agree with the operating system's file naming conventions.
- *datlen*
The record length of the file for fixed-length files. For variable-length files this is the size of the fixed-length portion of the record.
- *xtdsiz*
The file extension size. This is the amount by which the file is increased when additional space is needed.
- *filmod*
The c-tree Plus file mode used by your application. See Section 3.4.5 "File Modes" on c-tree Plus Programmer's Reference Guide for details.
- *num_idx*
The number of indices associated with this data file. Each such index will be either a separate c-tree Plus file or a member of an index file. See [CreateIndexFile\(\)](#) and [CreateIndexMember\(\)](#) on c-tree Plus Function Reference Guide.
- *# flds*
The number of fields contained in the data record, which should match the number of Field Description Records in your OTP file.
- *fst fldnam*
The symbolic name of the first field entered in the field section.
- *lst fldnam*
The symbolic name of the last field entered in the field section.

Field Description Record

field name	offset	data type	length
co_ordrdate	4	75	4

Parameters

field name

The symbolic name of the field used to identify a region of the data record.

offset

The physical offset from beginning of the record (base 0) at which this field starts.

data type

A numeric representation of the data type contained in this field, from the table below.

Data type values	Data type descriptions
16	1 byte signed character.
24	1 byte unsigned character.
33	2 byte signed integer.
41	2 byte unsigned integer.
51	4 byte signed integer.
59	4 byte unsigned integer.
67	4 byte integer representing money in pennies.
75	a date value represented as a 4 byte long.
83	a time value (seconds since midnight) as a long.
91	single float.
103	double float.
144	fixed length string.
146	null delimited varying length string.

length

The field length in bytes.

Index File Description Record

Each Index File record is comprised of eleven required fields, the first seven fields corresponding to parameters of **CreateIndexFile()**:

- index file name (*filnam*)
- key length (*keylen*)
- key type (*keytyp*)
- duplicate flag (*keydup*)
- number of additional index file members (*nomemb*)
- file size extension parameter (*xtdsiz*)
- file mode (*filmod*)
- NULL key flag (*nulkey*)
- empty character (*empchr*)
- number of key segments (*numseg*)
- symbolic index name (*symname*)

```

filnam      keylen keytyp keydup nomemb xtdsiz filmod nulkey empchr numseg      symname
custordr.idx 6      4      0      1      4096  1      1      32      1  custordr.1

```

Parameters

- *filnam*

The physical index file name without a path. The index file name must agree with the operating system's file naming conventions.

- *keylen*

The length of the key in this index.

- *keytyp*

The key type which can be:

0	fixed length key	8	padding compression
4	leading character compression	12	4 and 8 combined

- *keydup*

A value of 1 allows duplicates. A value of 0 does not allow duplicates. If duplicates are allowed, the last four bytes of the key value are reserved for the four bytes of the associated data record position. For example, a key length of twelve bytes combined with duplicate values results in eight bytes for the actual key value and four bytes for the tie-breaking data record position.

- *nomemb*

The number of additional index members contained in this index not counting the host index (physical file).

- *xtdsiz*

As with data files this is the amount by which the file will be extended when additional space is needed.

- *filmod*

c-tree Plus file mode used to open the file.

- *nulkey*

When set to 1, key values containing only the empty character, *empchr*, are excluded from the index. When set to 0 there is no check for NULL keys.

- *empchr*

The character determining if a key entry is a NULL key.

- *numseg*

The number of segments comprising the key. These are described below in the Key Segment Description Record.

- *symbolic name*

A symbolic name used to identify this particular index.

Optional Index Member Record

It is not necessary for each index associated with a data file to be in a separate file. Indices can be combined in the same physical file. If the *nomemb* parameter of an Index File Description Record is greater than zero, *nomemb* Index Member Records must follow the Index File Description Record. For example, if a data file has three indices, and if the second index has one additional member, (*nomemb* equals one), then there will be a total of two Index Description Records and one Index Member Record.

The Index Member Record is a subset of the Index File Description Record composed of seven required fields:

- key length (*keylen*)
- key type (*keytyp*)
- duplicate flag (*keydup*)
- NULL key flag (*nulkey*)
- empty character (*empchr*)

- number of key segments (*numseg*)
- symbolic index name (*symname*)

These fields have the same interpretation as the corresponding fields in the Index Description Record.

keylen	keytyp	keydup	nulkey	empchr	numseg	symname
8	4	1	1	32	1	custordr.2

Parameters

- *keylen*
The length of the key.
- *keytyp*
The type of the key as described for the host index above.
- *keydup*
1=allow duplicates, 0=do not allow duplicates.
- *nulkey*
1=check for NULL keys, 0=do not check for NULL keys.
- *empchr*
The character value used for the NULL key check.
- *numseg*
The number of segments comprising the key.
- *symbolic name*
The symbolic name to be used to identify this index.

Key Segment Description Record

Each key value is composed of one or more segments. A segment is simply a sequence of bytes from a specified offset within the data record. Each Key Segment Description record is comprised of three fields:

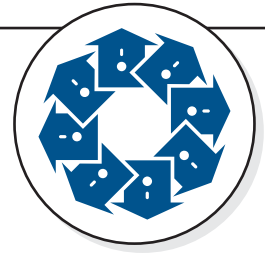
- segment position (*soffset*)
- segment length (*slength*)
- segment mode (*segmode*)

(soffset)	(slength)	(segmode)
19	4	2

Parameters

- *soffset*
The offset, in bytes, from the beginning of the record. When using segment types 4 or 5, represents a field number, starting at zero, in the variable-length portion of a variable-length record, not counting fields from the fixed-length portion.
- *slength*
The length of this segment of the key from 'offset'.
- *segmode*

The type of data that makes up this segment, as shown in the table in "Key Segment Modes" in *c-tree Plus Programmer's Reference Guide*.



ctree ODBC Driver c-tree Plus Edition

The c-tree® ODBC Driver - c-tree Plus® Edition was developed for c-tree Plus Standard and Extended files. This chapter describes the features unique to the c-tree ODBC Driver - c-tree Plus Edition.

2.1 Supported Data Types

The c-tree ODBC Driver supports the following data types. Additional types can be added with the c-tree Driver SDK described below.

If multiple c-tree Plus types relate to one SQL type, the default c-tree Plus type for that SQL type is in bold type. For example, the c-tree Plus types **CT_INT2** and **CT_INT2U** both map to **SQL_SMALLINT**. Going from SQL to c-tree Plus, **SQL_SMALLINT** maps to **CT_INT2**.

c-tree Plus Data Type			SQL Data Type	
Name	Keyword	Description	Name	Description
CT_ARRAY	LONGVARBINARY	Variable-length binary data with a 4-byte length	SQL_LONGVARBINARY	Binary data < 2 gigabyte
CT_BOOL	LOGICAL	One byte Boolean	SQL_BIT	1 byte. 1 is true. 0 is false.
CT_CHAR	TINYINT	Signed one-byte integer	SQL_TINYINT	1 byte unsigned number
CT_CHARU	TINYINT	Unsigned one-byte integer		
CT_INT2	SMALLINT	Signed two-byte integer	SQL_SMALLINT	2 byte integer
CT_INT2U	SMALLINT	Signed two-byte integer		
CT_INT4	INTEGER	Signed four-byte integer	SQL_INTEGER	4 byte integer
CT_INT4U	INTEGER	Unsigned four-byte integer		

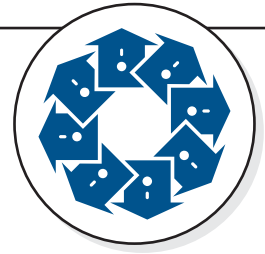
c-tree Plus Data Type			SQL Data Type	
CT_MONEY	DOUBLE	Signed four-byte integer interpreted as number of pennies.	SQL_DOUBLE	IEEE 8 byte float. Note that SQL_DOUBLE maps to CT_DFLOAT.
CT_DATE	DATE	Unsigned four-byte integer interpreted as the number of days since 02/28/1700.	SQL_DATE	IEEE 8 byte double.
CT_TIME	TIME	Unsigned four-byte integer interpreted as the number of seconds since midnight.	SQL_TIME	IEEE 8 byte double.
CT_TIMES	TIMESTAMP	Eight-byte floating point interpreted as the number of days since 12/30/1899 plus the number of seconds since midnight divided by 86400.	SQL_TIMESTAMP	IEEE 8 byte double.
CT_SFLOAT	REAL	Four-byte floating point	SQL_REAL	IEEE 4 byte float.
CT_DFLOAT	DOUBLE	Eight byte floating point	SQL_DOUBLE	IEEE 8 byte float
CT_SQLBCD	NUMERIC	Undefined in c-tree Plus. Use the c-tree Driver SDK to perform conversion for your application.	SQL_DECIMAL	Exact numeric quantity with up to 72 decimal digits of precision.
CT_FSTRING	CHAR	Fixed length field delimited data	SQL_CHAR SQL_LONGVARCHAR	Character data < 256 bytes Character data >= 256 bytes and < 2 gigabytes. Note that SQL_LONGVARCHAR maps to CT_STRING
CT_F2STRING	MEMO	Fixed length data with 2-byte length count.		
CT_F4STRING	MEMO	Fixed length data with 4-byte length count.		

c-tree Plus Data Type			SQL Data Type	
CT_STRING	VARCHAR	Varying length field delimited data	SQL_VARCHAR SQL_LONGVARCHAR	Character data < 256 bytes Character data >= 256 bytes and < 2 gigabytes
CT_2STRING	MEMO	Varying length data with 2-byte length count		
CT_4STRING	MEMO	Varying length data with 4-byte length count		
CT_PSTRING	MEMO	Varying length data with 1-byte length count		
CT_FPSTRING	CHAR	Fixed length data with 1-byte length count.	SQL_CHAR	Character data < 256 bytes Note that SQL_CHAR maps to CT_FSTRING

2.2 Suppression of ctree dialog boxes

A setting called DebugLog, which is a registry key for the 32-bit c-tree ODBC Driver and an *ODBC.INI* setting for the 16-bit Driver, indicates whether driver error messages should be displayed in dialog boxes or logged to *ctodbc.log*. The following string values are accepted for DebugLog:

Value	Meaning
Both	Display error in dialog box and log to <i>ctodbc.log</i> .
Screen	Display error in dialog box only.
File	Log to <i>ctodbc.log</i> only.
None	Do not log or display error messages.



ctree® Crystal Reports™ Driver

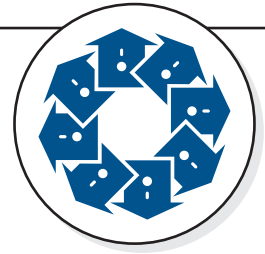
The c-tree Crystal Reports Driver allows Seagate Crystal Reports to directly access c-tree Plus® Standard and Extended files. This chapter describes the features unique to the c-tree Crystal Reports Driver.

3.1 Supported Data Types

The c-tree Crystal Reports Driver supports the following data types. Additional types can be added with the c-tree Driver Types SDK described below.

Crystal data type	c-tree Plus data type	Explanation of c-tree Plus data type
Boolean	CT_BOOL	One byte Boolean.
1 byte signed integer	CT_CHAR	Signed char.
1 byte unsigned integer	CT_CHARU	Unsigned char.
2 byte signed integer	CT_INT2	Signed two-byte integer.
2 byte unsigned integer	CT_INT2U	Unsigned two-byte integer.
4 byte signed integer	CT_INT4	Signed four-byte integer.
4 byte unsigned integer	CT_INT4U	Unsigned four-byte integer.
Currency	CT_MONEY	Signed four-byte integer interpreted as number of pennies.
Date	CT_DATE	Signed four-byte integer interpreted as the number of days since 02/28/1700.
Time	CT_TIME	Unsigned four-byte integer interpreted as the number of seconds since midnight.
Timestamp	CT_TIMES	Eight-byte floating point interpreted as the number of days since 12/30/1899 plus the number of seconds since midnight divided by 86400.
Double Precision Float	CT_SFLOAT	Four-byte floating point.
	CT_DFLOAT	Eight-byte floating point.

Crystal data type	c-tree Plus data type	Explanation of c-tree Plus data type
String fields	CT_FSTRING	Fixed length field delimited data.
	CT_FPSTRING	Fixed length data with 1-byte length count.
	CT_F2STRING	Fixed length data with 2-byte length count.
	CT_F4STRING	Fixed length data with 4-byte length count.
	CT_STRING	Varying length field delimited data.
	CT_PSTRING	Varying length data with 1-byte length count.
	CT_2STRING	Varying length data with 2-byte length count.
	CT_4STRING	Varying length data with 4-byte length count.



ctree ODBC Driver - c-tree V4 Edition

The c-tree® ODBC Driver was originally developed for files created with c-tree Plus®. The c-tree ODBC Driver - c-tree® V4 Edition supports only files created with c-tree V4.x. This section describes the features unique to the c-tree ODBC Driver - c-tree V4 Edition.

4.1 Requirements

The c-tree ODBC Driver - c-tree V4 Edition requires OTP files (described elsewhere). c-tree V4.x does not support resources, so there is no method for storing the required file and field definitions within the data files.

4.2 Supported Data Types

The c-tree ODBC Driver supports the following data types. Additional types can be added with the c-tree Driver SDK described below.

If multiple c-tree types relate to one SQL type, the default c-tree type for that SQL type is in bold type. For example, the c-tree types *CT_INT2* and *CT_INT2U* both map to *SQL_TINYINT*. Going from SQL to c-tree, *SQL_TINYINT* maps to *CT_INT2*.

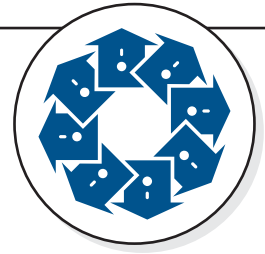
c-tree Data Type		SQL Data Type	
Name	Description	Name	Description
CT_ARRAY	Variable-length binary data with a 4-byte length.	SQL_LONGVARBINARY	Binary data < 2 gigabyte.
CT_BOOL	One byte Boolean.	SQL_BIT	1 byte. 1 is true. 0 is false.
CT_CHAR	Signed one-byte integer.	SQL_TINYINT	1 byte unsigned number.
CT_CHARU	Unsigned one-byte integer.		
CT_INT2	Signed two-byte integer.	SQL_SMALLINT	2 byte integer.
CT_INT2U	Unsigned two-byte integer.		

c-tree Data Type		SQL Data Type	
Name	Description	Name	Description
CT_INT4	Signed four-byte integer.	SQL_INTEGER	4 byte integer.
CT_INT4U	Unsigned four-byte integer.		
CT_MONEY	Signed four-byte integer interpreted as number of pennies.	SQL_DOUBLE	IEEE 8 byte float.
CT_DATE	Unsigned four-byte integer interpreted as the number of days since 02/28/1700.	SQL_DATE	IEEE 8 byte double.
CT_TIME	Unsigned four-byte integer interpreted as the number of seconds since midnight.	SQL_TIME	IEEE 8 byte double.
CT_TIMES	Eight-byte floating point interpreted as the number of days since 12/30/1899 plus the number of seconds since midnight divided by 86400.	SQL_TIMESTAMP	IEEE 8 byte double.
CT_SFLOAT	Four-byte floating point.	SQL_REAL	IEEE 4 byte float.
CT_DFLOA T	Eight byte floating point.	SQL_DOUBLE	IEEE 8 byte float.
CT_SQLBC D	Undefined in c-tree Plus. Use the c-tree Driver SDK to perform conversion for your application.	SQL_DECIMAL	Exact numeric quantity with up to 72 decimal digits of precision.
CT_FSTRIN G	Fixed length field delimited data.	SQL_CHAR SQL_LONGVARC HAR	Character data < 256 bytes. Character data 256 bytes and < 2 gigabytes. Note that SQL_LONGVARCHAR maps to CT_STRING
CT_F2STR ING	Fixed length data with 2-byte length count.		

c-tree Data Type		SQL Data Type	
Name	Description	Name	Description
CT_F4STRING	Fixed length data with 4-byte length count.		
CT_STRING	Varying length field delimited data.	SQL_VARCHAR SQL_LONGVARCHAR	Character data 256 bytes. Character data 256 bytes and < 2 gigabytes.
CT_2STRING	Varying length data with 2-byte length count.		
CT_4STRING	Varying length data with 4-byte length count.		
CT_PSTRING	Varying length data with 1-byte length count.		
CT_FPSTRING	Fixed length data with 1-byte length count.	SQL_CHAR	Character data < 256 bytes Note that SQL_CHAR maps to CT_FSTRING

4.3 c-tree V4 Driver with LONGVARD support off

The c-tree ODBC Driver - c-tree V4 Edition uses the LONGVARD #define to support variable-length records larger than 64K. However, some customers do not use this #define when building their c-tree library. These customers are not able to read variable-length data records using the current Driver. The Driver returns error 158 because the variable-length record header definition differs between LONGVARD and non-LONGVARD c-tree libraries. A c-tree V4 Edition ODBC Driver is available without LONGVARD support for these customers. Contact FairCom for details.



ctree Driver SDK

The c-tree Driver SDK provides the source code to all the necessary modules to control the data transfer between c-tree Plus and the c-tree Drivers, expanding the capabilities described in the c-tree Driver chapters and guides. The c-tree Driver SDK is documented here for informational purposes only. The c-tree Driver SDK must be purchased separately and requires a distribution license.

The c-tree Driver SDK gives you the ability to:

- Define special type mappings between c-tree Plus types and c-tree Driver types.
- Define data type capacities.
- Define your own appropriate field level padding.
- Intercept each column's data as it is transferred between c-tree Plus and the c-tree Driver.
- Intercept c-tree Plus records just before they are written to disk.
- Intercept c-tree Plus records just after they are read from disk.
- Handle special conversions between your data stored on disk and the c-tree Driver.
- Define a column's maximum capacity, scale and precision.

The c-tree Driver SDK is implemented through a custom DLL developed by you using the functions defined later to accomplish the tasks listed above. Make your custom DLL available to the c-tree Driver by placing the DLL name in the "Special Data Type Conversion DLL Name" field in the c-tree Driver Setup dialog. These notes provide the details for creating your custom DLL.

Throughout this chapter, the term "c-tree Driver" refers to any of the c-tree ODBC Driver - c-tree Plus Edition, the c-tree ODBC Driver - c-tree V4 Edition, and the c-tree Crystal Reports™ Driver.

5.1 Installation

The c-tree Driver SDK is distributed in a .ZIP file format containing everything needed to build your DLL except the compiler. This includes all the necessary c-tree Plus header files. These headers have been set to be compatible with the c-tree Drivers and must be used. Do NOT use any other c-tree Plus headers. The c-tree Driver SDK make files are set to point to these specific headers.

To install:

1. Create a work directory:

```
mkdir ODBCSDK
```

2. Unzip the SDK into this directory. Be sure to use the -d switch to extract all subdirectories:

```
pkunzip -d a:\odbcSDK.zip
```

3. This completes the installation process.

5.2 Example 1: Compiling and Checking Your Custom DLL

This section walks through building and verifying your custom DLL. All functions accessed by the c-tree Drivers are contained in a single C source module named *otcustm.c*.

In this example, we show how to make a small change to *otcustm.c*, compile it into a DLL, define it in the ODBC setup, and verify proper operation. The individual functions found in *otcustm.c* are detailed later.

1. Edit the file *source\otcustm.c*. At the top you will see the following define commented out:

```
// #define otGETIT_SAMPLE
```

Activate this #define. If you search the code for this #define, you will see it forces all long integers read from disk to be overridden with a hard-coded value of 4. After making this change, we expect any long integer read from a c-tree Plus record will be changed to the value 4 before it is displayed by application accessing the c-tree Driver.

2. Edit the make file. There are two make files provided, one for the 16 bit drivers, *otset16.mak*, and one for 32 bit drivers, *otset32.mak*. Edit the appropriate make and be sure the PATH is set to find your compiler correctly. Look for the following lines in the make:

```
# PATH Definition Section
coBAS =P:\msvc          # Compiler's Base
coBIN =P:\msvc\bin      # Compiler's BIN
coLIB =P:\msvc\lib      # Compiler's LIB
coINC =P:\msvc\include  # Compiler's INCLUDE
```

Specifically, the 32-bit compiler's base path will need to be changed for the following reason.

Hint: We set the compiler path within our makes to drive P:. Instead of changing the makes when going from machine to machine, we use the DOS **subst** command to establish drive P:. For example, if your compiler is on your drive D:, the following command executed from a command prompt will establish drive P: to point to your D: drive.

```
subst P: D:\
```

Note: Borland users must use the files provided in the BORLAND subdirectory instead of the default files.

3. Compile your DLL by executing *mk16.bat* for 16 bit or *mk.bat* for 32 bit.

```
mk32.bat
```

A DLL named *ot_usr16.dll* (16 bit) or *ot_usr32.dll* (32 bit) will be created in a LIB directory (*lib.16b* or *lib.32b*). Enter this DLL name in the DLL conversion name field in the c-tree Driver Setup.

4. Copy your DLL into the Windows System Directory (*system32* for NT). Execute the ODBC administrator and enter your DLL name (*ot_usr16.dll* or *ot_usr32.dll*) under the setup options "Special Data Type Conversion DLL Name" field. You may enter a fully qualified DLL name (e.g., *F:\my_odbc\lib.32b\ot_usr32.dll*) in setup and alleviate the need to copy the DLL to the Windows's system directory.

Execute an appropriate application for the Driver and use FairCom's sample table OTDEMO included with this SDK. If you view the Long column, you will see that all long integer data is forced to a value of 4. Repeat steps 1 through 4 again, this time deactivating the #define *otGETIT_SAMPLE*. Your long integer data will reflect what is in the c-tree Plus record.

5.3 Source Code Overview

The source code that handles all the capabilities offered by this SDK is contained in one primary file in the source directory: *otcustm.c*. The best way to master this kit's flexibility is to review the source code of this file. Each function contains documentation on its purpose. Reading this information, studying the code, and trying some examples is the best way to understand this SDK.

The fifteen functions affecting the Driver's operation can be categorized into four functions.

Primary data I/O Functions (read/write or get/set)

- **OT_SETIT()** - Set Column Data.
- **OT_GETIT()** - Get Column Data.

Type Definition Functions

- **OT_STMSP()** - Set Max, Scale, and Precision for a c-tree Plus type.
- **OTCtreeTypeToSqlType()** - Convert a c-tree Plus data type to an SQL data type.
- **OTSqLTypeToCtreeType()** - Convert an SQL data type to a c-tree Plus data type.

Type Verification Functions

- **OT_STFLEN()** - Set the *DODA's flen* based on type.
- **OT_CHKTYP()** - o-tree check type.

Optional (Sample) Functions

- **OT_GETIFIL()** - Retrieve *IFIL* for data file.
- **OT_GETDODA()** - Retrieve *DODA* for data file.
- **OT_SETDK()** - Intercept c-tree Plus data record just before written to disk.
- **OT_GETDK()** - Intercept c-tree Plus data record just after read from disk.
- **OT_FLDLEN()** - Schema Field Length Calculation.
- **OT_AlignAdjust()** - Alignment Adjust routine.
- **OT_Connect()** - Intercept a connection to the c-tree Driver.
- **OT_Disconnect()** - Intercept a disconnection from the c-tree Driver.

Note: You will also see the module *otrtree.c* in the source directory. This file contains the r-tree DATE type conversion routines to perform the default data conversion in *otcustm.c*.

5.4 Function Descriptions

OT_SETIT

Set Column Data

```
ctCONV LONG ctDECL OT_SETIT(pSchema, dptr, dlen, sptr, slen,
                             stype, padding, columnID, vfld)

pCTREE_SCHEMA pSchema; /* Pointer to schema information */
pVOID dptr; /* Destination buffer pointer (c-tree's Buffer) */
UCOUNT dlen; /* Destination length - (c-tree's doda->flen) */
void FAR *sptr; /* Source pointer (ODBC data buffer) */
ULONG slen; /* length of data in ODBC buffer */
COUNT stype; /* c-tree Type of data in ODBC data buffer */
NINT padding; /* c-tree padding character */
NINT columnId; /* Field number (first field=1) */
NINT vfld;
/* Field in variable length portion of ctree record flag */
```

The c-tree Driver calls this function to “set” the external data type into the c-tree Plus type. If a column’s data is inserted or modified, this function will be called passing in the source data and expecting this function to “set” it onto the c-tree Plus conversion buffer (destination). Here is where type conversion at the individual column level takes place from external sources to c-tree Plus. This function can be thought of as a form of copy, providing the source and the destination, whose job is to take the source and get it into the destination. Type conversion can take place in between. This function **MUST RETURN** the length of the data placed in the destination buffer that reflects the size of the c-tree Plus data being set. If type conversion is necessary, you must change the destination pointer (*dptr) to reflect your own converted destination work buffer. You will see this done in many places where FairCom uses ‘NewDataWorkBuffer’.

OT_GETIT

Get Column Data

```
ctCONV LONG ctDECL OT_GETIT(pSchema,tableType,dptr, dlen, sptr,
                             slen, flen, stype, columnId,
                             OEMtoANSIconv)

pCTREE_SCHEMA pSchema; /* Pointer to schema information */
COUNT tableType; /* ctVIRTUAL_TABLE=0 or ctREAL_TABLE=1 */
void FAR *dptr; /* Destination buffer (ODBC Buffer) */
ULONG dlen; /* Destination Buffer max Length */
void FAR *sptr; /* Source data buffer (c-tree's buffer) */
ULONG slen; /* Source data actual length */
ULONG flen; /* Source data field (DODA or field type) length */
COUNT stype; /* source date type (c-tree's type) */
NINT columnId; /* Field number (first field=1) */
COUNT OEMtoANSIconv; /* Convert OEM string data to ANSI */
```

The c-tree Driver calls this function to “get” the data from c-tree Plus into and external type. Each column ‘read’ calls this function passing in the c-tree Plus data (source) and expecting this function to “get” it onto the destination buffer (destination). Here is where type conversion at the individual column level takes place from c-tree Plus to external data types. This function can be thought of as a form of copy, providing the source and the destination and whose job is to take the source and get it into the destination. Type conversion can take place in between. This function **MUST RETURN** the length of the data placed in the destination buffer that relates directly to the external type. c-tree Plus (converted) data must be placed into the destination buffer, *dptr*, and never exceed the destination buffer’s length, *dlen*.

OT_STMSP

Set Max, Scale, and Precision for a c-tree Plus type

```
ctCONV void ctDECL OT_STMSP(dodaptr, colMax, colScale,
                             colPrecision)

pDATOBJ dodaptr; /* DODA pointer */
pULONG colMax; /* Column MAX Length */
pCOUNT colScale; /* Column Scale */
pCOUNT colPrecision; /* Column Precision */
```

This function must reflect the maximum column length for c-tree Plus types. It may also be used to define the scale and precision for special BCD types.

OTCtreeTypeToSqlType

Convert c-tree Plus type to SQL type

```
ctCONV ULONG ctDECL OTCtreeTypeToSqlType(pDatObj)
```

```
pDATOBJ pDatObj; /* Pointer to information for this field */
```

This is where the definition of type mapping occurs between c-tree Plus types and external types.

OTSqlTypeToCtreeType

Convert SQL type to c-tree Plus type

```
ctCONV NINT ctDECL OTSqlTypeToCtreeType(SqlType)
```

```
ULONG SqlType; /* SQL data type */
```

This is where the definition of type mapping occurs between external types and c-tree Plus types.

OT_GETIFIL

Retrieve IFIL for data file

```
ctCONV VRLen ctDECL OT_GETIFIL(tablenam, bufsiz, bufptr,
                               retval, errcode)
```

```
pTEXT    tablenam;
LONG     bufsiz;
pVOID    bufptr;
VRLen    retval;
COUNT   errcode;
```

The c-tree Driver calls **OT_GETIFIL()** to retrieve the *IFIL* for the current data file. Prior to calling **OT_GETIFIL()**, the c-tree Driver calls the c-tree Plus **GetFile()** function to attempt to retrieve the *IFIL* from the data file. The return code from **GetFile()** is passed to **OT_GETIFIL()** as *retval* and the value of *isam_err* following the **GetFile()** call is passed to **OT_GETIFIL()** as *errcode*. The c-tree Driver also passes in the name of the data file (*tablenam*), a return buffer (*bufptr*) containing the *IFIL* read from the data file (if **GetFile()** found an *IFIL* in the data file), and the buffer size (*bufsiz*). Use **OT_GETIFIL()** to modify existing *IFIL* information or to create an entirely new *IFIL* if one does not already exist for the given data file.

The c-tree Driver makes the following calls to **OT_GETIFIL()**:

```
/* Get the size of the IFIL, and allocate a buffer large enough to hold the IFIL. */
ifilSize = OT_GETIFIL(tablenam, filno, 0L, (pVOID)0);
```

```
/* Retrieve the IFIL into the buffer allocated by the c-tree Driver. */
OT_GETIFIL(tablenam, filno, ifilSize, ifilPtr);
```

See the code surrounded by `#ifdef otRESOURCE_SAMPLE` in *otcustm.c* for one way to use **OT_GETIFIL()** (to create an *IFIL* for a file not containing one).

OT_GETDODA

Retrieve DODA for data file.

```
typedef ctCONV VRLen (ctDECL *pfGETDODA)(COUNT datno,
                                         LONG bufsiz, pVOID bufptr, COUNT mode);
```

```
ctCONV VRLen ctDECL OT_GETDODA(GETDODA, pSchema, ifil, bufsiz,
                               bufptr, mode)
```

```
pfGETDODA    GETDODA; /* typedef'ed above */
pCTREE_SCHEMA pSchema;
pIFIL        ifil;
LONG         bufsiz;
pVOID        bufptr;
COUNT       mode;
```

The c-tree Driver calls **OT_GETDODA()** to retrieve the *DODA* for the current data file. The c-tree Driver passes in a function pointer to c-tree Plus's **GetDODA()** routine, a pointer to the c-tree Plus schema information, the size of the buffer passed in, a pointer to a buffer in which the *DODA* is to be returned, and the mode to be used in calling **GetDODA()**.

The c-tree Driver makes the following calls to **OT_GETDODA()**:

```
/* Get the size of the schema map and allocate a buffer large enough to hold the schema map. */
schemaSize = OT_GETDODA(GETDODA, NULL, IFILptr, 0L, (pVOID)0, SCHEMA_MAP);

/* Retrieve the schema map into the buffer allocated by the c-tree Driver. */
OT_GETDODA(GETDODA, NULL, IFILptr, schemaSize, schemaPtr, SCHEMA_MAP);

/* Get the size of the DODA and allocate a buffer large enough to hold the DODA. */
dodaSize = OT_GETDODA(GETDODA, NULL, IFILptr, 0L, (pVOID)0, SCHEMA_DODA);

/* Retrieve the DODA into the buffer allocated by the c-tree Driver. */
OT_GETDODA(GETDODA, pSchema, IFILptr, dodaSize, dodaPtr, SCHEMA_DODA);
```

See the code surrounded by **#ifdef otGETDODA_SAMPLE** in *otcustm.c* for one way to use **OT_GETDODA()** (to alter the data types for fields in the *DODA*).

OT_SETDK

Intercept c-tree Plus data record just before written to disk

```
ctCONV COUNT ctDECL OT_SETDK(pSchema, pBuffer, BufferSize,
                             pUsedBufferSize)
```

```
pCTREE_SCHEMA    pSchema;
/* Pointer to current c-tree Plus ODBC schema info */
pVOID            pBuffer;
/* Record buffer about to be written to disk */
pVRLEN          BufferSize;
/* Total size of record buffer to be written to disk */
pVRLEN          pUsedBufferSize;
/* Actual size of record buffer to be written to disk */
```

This function is called just before a c-tree Plus **ADD()** or **REWRITE()** function. It provides a means for you to intercept the c-tree Plus record on its way to disk. This function should only be implemented by advanced c-tree Plus users with caution. Contained within is some sample code to decipher the c-tree Plus record buffer.

OT_GETDK

Intercept c-tree Plus data record just after read from disk

```
ctCONV COUNT ctDECL OT_GETDK(pSchema)
```

```
pCTREE_SCHEMA pSchema;
/* Pointer to current c-tree Plus ODBC schema info */
```

This function is called just after a successful read from disk of a c-tree Plus record. It provides a means for you to intercept the c-tree Plus record just after the read, and before it is processed by the c-tree Driver. This function should only be implemented by advanced c-tree Plus users with caution. See the **CTREE_SCHEMA** definition below for the information provided to you by *pSchema*.

```
typedef struct {
pDATOBJ pDatObj; /* Pointer to DODA */
pVOID pBuffer; /* Pointer to record buffer going to disk */
VRLEN bufferSize; /* Total size of record buffer-amount allocated */
VRLEN dodaSize; /* size of the doda */
UINT align; /* alignment adjustment factor */
NINT delimiter; /* field delimiter character from file's header */
NINT padding; /* field padding character from file's header */
}
```

```

UINT    nfields;    /* number of fields in record (in DODA)          */
UCOUNT  vlength;   /* byte offset of variable record portion- 0=fixed length */
VRLEN   UsedbufferSize;
        /* Used size of record buffer for variable length records */
NINT    lastfield; /* last field updated for current record          */
pVOID   pUsrData;  /* User data pointer                                */
} CTREE_SCHEMA, ctMEM *pCTREE_SCHEMA,
  ctMEM * ctMEM *ppCTREE_SCHEMA;

```

OT_STFLEN

Sets the DODA's flen based on type

```
ctCONV COUNT ctDECL OT_STFLEN(dodaptr,colMax)
```

```

pDATOBJ dodaptr; /* DODA Pointer          */
ULONG   colMax;  /* Max Column width */

```

This function is used by the c-tree Driver to verify a *DODA's flen* value based on its c-tree Plus type. It is used for internal *DODA* verification.

OT_CHKTYP

o-tree check type

```
ctCONV COUNT ctDECL OT_CHKTYP(dodaptr,dodaTypeName)
```

```

pDATOBJ dodaptr; /* DODA pointer */
pTEXT   dodaTypeName; /* Output Text */

```

This function is used when c-tree Plus debug is turned on. Verifies the integrity of a *DODA*. It checks a file's *DODA's flen* value against what FairCom thinks it should be. It also sets the output text for the *ctodbc.log* file for when a *DODA* is dumped to the log.

OT_FLDLEN

Schema Field Length Calculation

```
COUNT OT_FLDLEN(pSchema,pDatObj,pField,pFieldLen,pActualLen)
```

```

pCTREE_SCHEMA pSchema; /* Pointer to the CTREE_SCHEMA structure */
pDATOBJ       pDatObj; /* Pointer to the DODA          */
pVOID         pField;  /* Pointer to the current field's data */
pVRLEN       pFieldLen; /* Return total field length (total buffer) */
pVRLEN       pActualLen; /* Return actual field length (used in buffer) */

```

Part of the sample code provided for the [OT_SETDK/OTGETDK\(\)](#) functions. It is used to calculate field lengths based on a field's type. It is used to help walk through the c-tree Plus record buffer. See [OT_SETDK\(\)](#) above.

OT_AlignAdjust

Alignment Adjust routine

```
UINT OT_AlignAdjust (alignment, fkind, fadr)
```

```

UINT alignment; /* alignment flag */
UINT fkind;     /* field type      */
pVOID fadr;     /* field address   */

```

Part of the sample code provided for the **OT_SETDK/OTGETDK()** functions. It is used to calculate alignment in order to determine the offset of a field's data. It is used to help walk through the c-tree Plus record buffer. See **OT_SETDK()** above.

OT_Connect

Connection routine

```
ctCONV COUNT ctDECL OT_Connect(ctcepfnc, ctWNGV)

pCEPFNC ctcepfnc; /* A single-entry point function pointer */
pCTGVAR ctWNGV; /* c-tree Plus global pointer */
```

Intercepts as an application connects to the c-tree Driver to execute the function referenced by *ctcepfnc*.

OT_Disconnect

Disconnection routine

```
ctCONV COUNT ctDECL OT_Disconnect(void)
```

Intercepts as an application disconnects from the c-tree Driver.

5.5 Example 2: Manipulating Data Read from a File

In "[Example 1: Compiling and Checking Your Custom DLL¹](#)", we enabled the `otGETIT_SAMPLE` `#define`, which set all long integers read from disk to a value of 4. That example demonstrated the use of **OT_GETIT()** to manipulate data read from a c-tree Plus data file. Now we will look at an example of using **OT_SETIT()** to manipulate data that will be written to a c-tree Plus data file.

Edit `source\otcustm.c` to enable the following `#define` and recompile your custom DLL:

```
// #define otSETIT_SAMPLE
```

This change converts any string field data passed to **OT_SETIT()** to uppercase. Use FairCom's sample table `OTDEMO` again and view the Description column. This is a string field using the c-tree Plus `CT_STRING` data type. Add a new entry and you will see the description field has been converted to uppercase upon viewing the new record. Notice the difference between this example and the first one. In "[Example 1: Compiling and Checking Your Custom DLL²](#)", we did not change the data written to the c-tree Plus data file, we simply forced the value of long integers to 4 upon retrieval from the data file. Therefore, long integers in both existing c-tree Plus data records and any records subsequently added appeared as a 4, but the underlying data in the c-tree Plus data file was not changed. In this example, no existing string data is affected, but string data in new records or old records that are updated is converted to uppercase. We modify data from the application before it is stored in the data file. This highlights the difference between **OT_GETIT()** and **OT_SETIT()**. Changes made to data in **OT_GETIT()** affect the data seen by applications, but the c-tree Plus data itself is not touched. Changes made to

¹ This section walks through building and verifying your custom DLL. All functions accessed by the c-tree Drivers are contained in a single C source module named `otcustm.c`. In this example, we show how to make a small change to `otcustm.c`, compile it into a DLL, define it in the ODBC setup, and verify proper operation. The individual functions found in `otcustm.c` are detailed later. Edit the file `source\otcustm.c`. At the top you will see the following define commented out: `// #define otGETIT_SAMPLEA ...`

² This section walks through building and verifying your custom DLL. All functions accessed by the c-tree Drivers are contained in a single C source module named `otcustm.c`. In this example, we show how to make a small change to `otcustm.c`, compile it into a DLL, define it in the ODBC setup, and verify proper operation. The individual functions found in `otcustm.c` are detailed later. Edit the file `source\otcustm.c`. At the top you will see the following define commented out: `// #define otGETIT_SAMPLEA ...`

data in **OT_SETIT()** will affect data stored to the c-tree Plus data file after the change has been made, but will not affect existing data unless the data is modified since the updated data will pass through **OT_SETIT()** before being updated in the c-tree Plus data file.

5.6 Example 3: Altering Data

Finally, here is an advanced example highlighting the following capabilities of the c-tree Driver SDK:

- Altering data read from the c-tree Plus data file before it is seen by applications.
- Altering data from an application writing to the c-tree Plus data file.
- Altering the interpretation of the data by the application.

To follow this example, enable the following `#define` in `source\otcustm.c`, and recompile your custom data types DLL:

```
// #define otSHOW_NATIVE_FORMAT
```

You may have noticed that, although we manipulated the content of the data in the first two examples, we did not alter the underlying data type seen by the application. In [“Example 1: Compiling and Checking Your Custom DLL³”](#), long integers were forced to have a value of 4 but were still seen as long integers. In [“Example 2: Manipulating Data Read from a File”](#), strings were forced to uppercase but were still strings. In this example, however, we remap a c-tree Plus data type to a different SQL data type from the default used by the c-tree Drivers.

By default, the c-tree Drivers map the c-tree Plus types `CT_DATE`, `CT_TIME`, and `CT_TIMES` to SQL data types `SQL_DATE`, `SQL_TIME`, and `SQL_TIMESTAMP` respectively. Each of these data types has its own physical representation.

`CT_DATE` and `CT_TIME` are stored as long integers and `CT_TIMES` is stored as a double in c-tree Plus data files. `SQL_DATE`, `SQL_TIME`, and `SQL_TIMESTAMP` values are stored in memory for use by ODBC-compliant applications as `SQL_DATE`, `SQL_TIME`, and `SQL_TIMESTAMP` structures. See `source\otcustm.h` for the layout of these structures.

By default, the c-tree Drivers handle the conversion of data between these physical representations. **OT_SETIT()** takes `SQL_DATE`, `SQL_TIME`, and `SQL_TIMESTAMP` structure data and converts it to long integer, long integer, and double data respectively. **OT_GETIT()** reverses this process. This example simply disables these conversions and tells applications that `CT_DATE` and `CT_TIME` data is now `SQL_INTEGER` data instead of `SQL_DATE` and `SQL_TIME` data and `CT_TIMES` data is now `SQL_DOUBLE` data.

Here is what was done to accomplish this type remapping. See the code surrounded by `#ifdef otSHOW_NATIVE_FORMAT` in `source\otcustm.c` for details:

- In **OT_SETIT()**, disable the conversion of `SQL_DATE`, `SQL_TIME`, and `SQL_TIMESTAMP` structured data into long integer, long integer, and double data when storing data from ODBC-compliant applications to c-tree Plus data files.
- In **OT_GETIT()**, disable the conversion of `CT_DATE`, `CT_TIME`, and `CT_TIMES` data into `SQL_DATE`, `SQL_TIME`, and `SQL_TIMESTAMP` format when reading data from c-tree Plus data files for use by ODBC-compliant applications.
- In **OTCtreeTypeToSqlType()**, return `SQL_INTEGER` for `CT_DATE` and `CT_TIME` data types and `SQL_DOUBLE` for the `CT_TIMES` data type to change the interpretation of these data types by ODBC-compliant applications.

³ This section walks through building and verifying your custom DLL. All functions accessed by the c-tree Drivers are contained in a single C source module named `otcustm.c`. In this example, we show how to make a small change to `otcustm.c`, compile it into a DLL, define it in the ODBC setup, and verify proper operation. The individual functions found in `otcustm.c` are detailed later. Edit the file `source\otcustm.c`. At the top you will see the following define commented out: `// #define otGETIT_SAMPLEA ...`

- In `OT_STMSP()`, return `sizeof(LONG)` for `CT_DATE` and `CT_TIME` data types and `sizeof(double)` for the `CT_TIMES` data type so that ODBC-compliant applications know the proper physical size of the remapped data types.

You can view the Date, Time, and Timestamp fields in the OTDEMO sample table to verify that the `CT_DATE`, `CT_TIME`, and `CT_TIMES` c-tree Plus data types have been successfully remapped as described above.

5.7 Debugging

This section describes how to debug a custom c-tree Driver DLL.

To step through custom c-tree Driver DLL code using the Microsoft compiler, first add the necessary debug flags to the compiler (`-Zi -Od`) and link (`-debug:full -debugtype:cv`) options in `otset32.mak`:

```
LINK = $(coBIN)\link -debug:full -debugtype:cv  
CFLAG2 = -c -DWIN32 -Zp1 -G3 -Gf -DotFOR_DLL -Zi -Od
```

After building the custom DLL, do the following:

1. Start up the Microsoft Visual Studio.
2. Select File | Open Workspace...
3. Browse for the ODBC-compliant application that you want to run (for example, `msqry32.exe` if using Microsoft Query).
4. Select Project | Settings... and click on the Debug tab.
5. In the Category drop-down list, select "Additional DLLs", and enter the full path and name of your custom DLL (`ot_usr32.dll`) in the list of modules. Click OK to save the change.
6. Open `otcustm.c` (using File | Open...), and set breakpoints as desired.
7. Press F5 to run the ODBC application. The application will break at the specified breakpoints.

Index

A

Alignment	
data	3
Arrays	
IFIL	1
IIDX	1
ISEG	1

C

Check custom DLL	22
Compile custom DLL	22
ctree Driver SDK	3, 21
c-tree Driver SDK	2
c-tree Drivers	1
ctree ODBC Driver - c-tree V4 Edition	17
ctree ODBC Driver c-tree Plus Edition	11
c-tree V4 Driver with LONGVARD support off	19
ctree 3 Crystal Reports_ Driver	15

D

Data	
alignment	3
conversion	2
file description record	6
float type	4
padding	3
SDK types	2
string lengths	4
supported types, all drivers	4
supported types, Crystal Reports	15
supported types, ODBC c-tree Plus	11
supported types, ODBC c-tree V4 edition	17
Data Alignment/Padding	3
Data dictionary	
predefined FairCom script	2
Data dictionary path	
debug info	5
Data File Description Record	6
Debug	5
custom DLL	30
enable	5
Debugging	30
Debugging Options	5
Dialog box	
setup debug	5
supress	13
DLL	
check	22
compile	22
DODA	1, 4
Driver	
choose	1

Crystal Reports	15
c-tree Driver SDK	2
distribution	2
ODBC c-tree Plus edition	11
ODBC c-tree V4 edition	17
requirements	1
SDK	21
Driver Distribution	2

E

Example 1	
Compiling and Checking Your Custom DLL	22
Example 2	
Manipulating Data Read from a File	28, 29
Example 3	
Altering Data	29

F

FAIRCOM TYPOGRAPHICAL CONVENTIONS	v
Field definitions	1
Field description record	7
Field Description Record	7
Field options	4
hidden	4
precision	4
read-only	4
read-only on update	4
required	4
scale	4
Field Options	4
File	
data description record	6
incremental ISAM definitions	1
OTP	5
vendor.db	2
Function Descriptions	23
Functions	
descriptions	23
optional/sample	23
primary data I/O	23
type definition	23
type verification	23

I

IFIL	1
IIDX	1
Index file description record	8
Index File Description Record	8
Index member record	9
Installation	21
driver SDK	21
ISEG	1
Items for Consideration	1

K

Key segment description record	10
Key Segment Description Record	10

L		
LONGVARD support off	19	
N		
Notes from your Software Provider	2	
O		
Optional (Sample) Functions	23	
Optional Index Member Record	9	
Options		
debug	5	
field	4	
OT_AlignAdjust	27	
OT_CHKTYP	27	
OT_Connect	28	
OT_Disconnect	28	
OT_FLDLEN	27	
OT_GETDK	26	
OT_GETDODA	25	
OT_GETIFIL	25	
OT_GETIT	24	
OT_SETDK	26	
OT_SETIT	23	
OT_STFLEN	27	
OT_STMSP	24	
OTCtreeTypeToSqlType	24	
OTP File Contents	6	
OTP File Layout	6	
OTP File Organization	6	
OTP File Requirements	5	
OTP files	5	
contents	6	
layout	6	
organization	6	
requirement	5	
OTP Files	1, 5	
OTSqlTypeToCtreeType	25	
P		
Padding		
data	3	
Primary data I/O Functions (read/write or get/set)	23	
R		
Record		
data file description	6	
field description	7	
index file description	8	
key segment description	10	
optional index member	9	
schema	1	
Requirements	17	
all drivers	1	
ODBC c-tree V4 edition	17	
S		
SDK		
		check DLL
		22
		compile DLL
		22
		install
		21
		source code
		22
		Source code
		22
		optional functions
		23
		primary data I/O functions
		23
		type definition functions
		23
		type verification functions
		23
		Source Code Overview
		22
		Supported data types
		4
		Crystal Reports
		15
		ODBC c-tree Plus edition
		11
		ODBC c-tree V4 edition
		17
		Supported Data Types
		4, 11, 15, 17
		Suppress dialog box
		13
		Suppression of ctree dialog boxes
		13
		T
		Type Definition Functions
		23
		Type Verification Functions
		23
		V
		VENDOR.DB
		2