CA-Clipper®

For DOS

Version 5.3

Drivers Guide

June 1995

COMPUTER ASSOCIATES®
Software superior by design.

Contents

Chapter 3: DBFCDX Driver Installation and Usage

Chapter 4: DBFMDX Driver Installation and Usage

Chapter 5: DBFNDX Driver Installation and Usage

Chapter 6: DBFNTX Driver Installation and Usage

Chapter 1 Introduction

Important! *Some of the topics in this guide are intended for advanced CA-Clipper developers. Much of this information is presented at a fairly high level and requires programming knowledge beyond the CA-Clipper language. Other parts are useful to users of all levels. Refer to the User Interface Levels section of the "Replaceable Database Driver Architecture" chapter to determine which part of the language is appropriate to your level of expertise.*

Based on your own level of experience with the CA-Clipper language, you should decide whether you wish to take advantage of these new and advanced features. The Reference Guide contains all existing CA-Clipper command and function syntax and descriptions, whereas this guide addresses only the new extensions to CA-Clipper. Understanding this information should enable you to increase the power and effectiveness of your applications.

CA-Clipper 5.3 supports a driver architecture that allows CA-Clipper-compiled applications to use replaceable database and terminal drivers. This *Drivers Guide* contains all the information you need to use the replaceable drivers provided as part of the CA-Clipper Development System.

Overview of RDD System

RDD is an abbreviation for Replaceable Database Driver, and it is used to describe an interface that controls how your application accesses and manipulates database and ancillary files.

CA-Clipper provides several RDDs to give you access to the database, memo, and index file formats of many popular database software products. By simply linking the proper RDD with your application, you get automatic, easy access to files created by other database engines.

Moreover, CA-Clipper gives you new and enhanced commands and functions designed to make your applications independent of the RDD in use. Using RDDs, you can give end users more flexibility in choosing to migrate to your CA-Clipper applications without losing data and to easily move their data back and forth between applications if they prefer.

Overview of Alternate Terminal Drivers

An *alternate terminal driver* is a library (.LIB) file that controls how your application addresses the screen output device. CA-Clipper provides several alternate terminal drivers to allow your applications to run in a wider variety of environments.

Note: To perform normal information presented in screen input/output in a CA-Clipper application, you do not need the *Drivers Guide.* The default database and terminal drivers are automatically linked and the commands and functions used for these purposes are discussed in the *Reference Guide.* For several categorized lists of these commands and functions, refer to "Appendix G: Categorized Language Tables" in the *Error Messages and Appendices Guide.*

In This Guide

This guide consists of nine chapters including this Introduction chapter.

For an online version of this guide accessible while operating your program editor or any other development utility, use *The Guide To CA-Clipper. The Guide To CA-Clipper* is an online documentation system that uses the Norton Instant Access Engine™. It is documented in the *Programming and Utilities Guide.*

Note: The Workbench provides a separate online help system for all menu commands, dialog boxes, and procedural steps. From the Workbench, press F1 or click on the Help pull-down menu to access this online help system.

Chapter 2: Replaceable Database Driver Architecture

The CA-Clipper database system supports a driver architecture that makes CA-Clipper-compiled applications data format independent. Such applications can, therefore, access the data formats of other database systems, including the dBASE IV (.mdx) and FoxPro (.cdx) formats on a variety of equipment. This chapter discusses how RDDs fit into the overall CA-Clipper architecture, defines the basic terminology you will need to understand subsequent chapters, and summarizes new and enhanced commands and functions designed to support the RDD architecture.

Chapter 3: DBFCDX Driver Installation and Usage

The DBFCDX database driver provides access to FoxPro 2 (.cdx) and (.idx) file formats. This chapter explains how to install DBFCDX and how to use it in your applications.

Chapter 4: DBFMDX Driver Installation and Usage

The DBFMDX database driver provides access to dBASE IV (.dbf), (.mdx), and (.dbt) file formats. The driver also supports dBASE IV-compatible file and record locking schemes, allowing shared access between CA-Clipper and dBASE IV programs. This chapter explains how to install DBFMDX and how to use it in your applications.

Chapter 5: DBFNDX Driver Installation and Usage

The DBFNDX database driver uses the CA-Clipper driver architecture to access dBASE III PLUS compatible index files within a CA-Clipper program, allowing you to create, access, and update dBASE III and dBASE III PLUS compatible index (.ndx) files. This chapter explains how to install DBFNDX and how to use it in your applications.

Chapter 6: DBFNTX Driver Installation and Usage

DBFNTX is the default database driver for CA-Clipper that lets you create and maintain index (.ntx) files with features above and beyond those supplied with previous versions of DBFNTX. This chapter details these new features and explains how to install and use DBFNTX in your applications.

Chapter 7: DBFMEMO Driver Installation and Usage

The DBFMEMO database driver provides greater flexibility when working with memo fields. This chapter explains how to install DBFMEMO and how to use it in your applications.

Chapter 8: DBFBLOB Driver Installation and Usage

The DBFBLOB database driver provides an alternate mechanism for working with memo fields. This chapter explains how to install DBFBLOB and how to use it in your applications.

Chapter 9: Alternate Terminal Drivers

CA-Clipper provides several alternate terminal drivers to allow your applications to run in a wider variety of environments. This chapter discusses how alternate terminal drivers fit into the overall CA-Clipper architecture, as well as how to install and use each of the supplied terminal drivers: ANSITERM, NOVTERM, and PCBIOS.

Chapter 2 Replaceable Database Driver Architecture

In This Chapter

CA-Clipper supports a driver architecture that allows CA-Clipper applications to use Replaceable Database Drivers (RDDs). The RDD system makes CA-Clipper applications data-format independent. Such applications can, therefore, access the data formats of other database systems, including the dBASE IV (.mdx) and FoxPro (.cdx) formats on a variety of equipment. This driver architecture can even support database drivers that are not file-based, although all of the drivers supplied with CA-Clipper 5.3 are file-based.

The concept of replaceable drivers is not new to this version of CA-Clipper. In previous versions, the use of the default database driver (DBFNTX.LIB) was hidden by the fact that it was automatically linked into your application. In fact, this is still the case. The DBFNTX driver has been replaceable since it was first introduced in version 5.0. Before this version, the DBFNTX driver was the only RDD supplied as part of the system.

With the introduction of the new RDDs, CA-Clipper provides many new and enhanced commands and functions that access and manipulate databases. These language elements can enable your applications to access data regardless of the RDD under which it is ordered. There are also commands and functions that give you specific information about the RDDs in use.

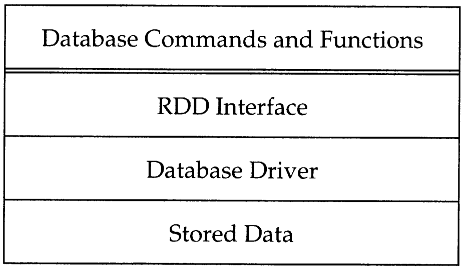
The Language Implementation section of this chapter includes tables that summarize these new and enhanced language elements. This chapter also covers basic terminology, implementation principals, and general concepts of the Order Management system.

The following major topics are discussed:

- RDD Basics
- Basic Terminology
- Language Implementation
- Order Management System

RDD Basics

The cornerstone of the replaceable database driver system is the CA-Clipper work area. All CA-Clipper database commands and functions operate in a work area through a database driver that actually performs the access to the stored database information. The layering of the system looks like this:



In this system, each work area is associated with a single database driver. Each database driver, in turn, is supplied as a separate library (.LIB) file that you link into your application programs. Within an application, you specify the name of the database driver when you open or access a database file or table with the USE command or DBUSEAREA() function. If you specify no database driver at the time a file is opened, the default driver is used. You may select which driver will be used as the default driver.

Once you open a database in a work area, the RDD used for that work area is automatically used for all operations on that database (except commands and functions that create a new table). Any command or function that creates a new table (i.e., SORT, CREATE FROM, DBCREATE(), etc.) uses the default RDD. Most of the new commands and functions let you specify a driver other than the default driver.

The normal default database driver, DBFNTX (which supports the traditional .dbf, .ntx, and .dbt files), is installed in your \CLIP53\LIB directory. This driver is linked into each program automatically to provide backwards compatibility.

To use any of the other supplied drivers, either as an additional driver or an alternate driver, you must use the REQUEST command to ensure that the driver will be linked in. You must also include the appropriate library on the link line.

All CA-Clipper applications will automatically include code generated by Rddsys.prg from the \CLIP53\SOURCE\SYS subdirectory. If you wish to automatically load another RDD, you must modify and compile Rddsys.prg and link the resulting object file into your application. The content of the default Rddsys.prg is shown below. Only the portion in **bold** should be modified.

```
//  Current Rddsys.prg

ANNOUNCE RDDSYS                     // This line must not change
INIT PROCEDURE RddInit
   REQUEST DBFNTX                   // Force link for DBFNTX RDD
   RDDSETDEFAULT( "DBFNTX" )        // Set up DBFNTX as default
                                    // driver

   RETURN

// eof: Rddsys.prg
```

To change the default to a new automatically-loading driver, modify the **bold** lines in Rddsys.prg to include the name of the new driver. For example:

```
//  Revised Rddsys.prg
#include "rddsys.ch"

ANNOUNCE RDDSYS                     // This line must not change
INIT PROCEDURE RddInit
   REQUEST DBFCDX                   // Force link for DBFCDX RDD
   RDDSETDEFAULT( "DBFCDX" )        // Set up DBFCDX as default
                                    // driver

   RETURN

// eof: Rddsys.prg
```

If you change this file, all CA-Clipper applications in which it is linked will automatically include the new RDD.

To use any RDD other than the default, you must explicitly identify it through the use of the VIA clause of the USE command.

You need not disable the automatic loading of DBFNTX in order to use other RDDs in your applications, but if your application will not use any DBFNTX functionality, you can save its code overhead by disabling it.

To completely disable the automatic loading of a default RDD, remove the two lines shown above in bold. For example:

```
//  New Revised Rddsys.prg
//  disables auto-loading
#include "rddsys.ch"

ANNOUNCE RDDSYS                    // This line must not change
INIT PROCEDURE RddInit

   RETURN
// eof: Rddsys.prg
```

Basic Terminology

The RDD architecture introduces several new terms and concepts that are key to the design and usage of RDDs. You should familiarize yourself with these concepts and terms as you begin to use the RDD functionality. The meaning of some earlier terminology is also further defined. The following RDD functional glossary defines the terminology for all RDDs.

- *Key Expression:* A valid CA-Clipper expression that creates a key value from a single record.
- *Key Value:* A value that is based on value(s) contained within database fields, associated with a particular record in a database.
- *Identity:* A unique value guaranteed by the structure of the data file to reference a specific record in a database even if the record is empty. In the Xbase file (.dbf), the identity is the record number; but it could be the value of a unique primary key or even the offset of an array in memory.
- *Keyed Pair:* A pair consisting of a key value and an identity.
- *Identity Order:* Describes a database arranged by identity. In Xbase, this refers to the physical arrangement of the records in the database in the order in which they were entered (natural order).
- *Tag:* A set of keyed pairs that provides ordered access to the table based on a key value. Usually, an order in a multiple-order index (order).

- *Order:* A named mechanism (index) that provides logical access to a database according to the keyed pairs. This term encompasses both single indices and the tags in multiple-tag indices.

  Orders are not, themselves, data files. They provide access to data that gives the appearance of an ordering of the data in a specific way. This ordering is defined by the relationships between keyed pairs. An order does not change the physical (the natural or entry) order of data in a database.

- *Controlling Order:* The active order (index) for a particular work area. Only one order may control a work area at any time, and it controls the order in which the database is accessed during paging and searching.

- *Order List:* A list of all the orders available to the database in the specified work area.

- *Order Bag:* A container that holds zero or more orders. Normally a disk or memory file. A traditional index like .ntx is an order bag that holds only one order. A multiple-tag index (.mdx or .cdx) is an order bag that holds zero or more orders. Though order bags may be a memory or disk file, CA-Clipper 5.3 only supports order bags as disk files.

- *Record:* A record in the traditional database paradigm is a row of one or more related columns (fields) of data. In the expanded architecture of CA-Clipper, a record could be data that does not exactly fit this definition.

  A record is, in this expanded context, data associated with a single *identity*. In an Xbase data structure, this corresponds to a row (fields associated with a record number); in other data structures, this may not be the case.

  In this document we use "record" in the traditional sense, but you should be aware that CA-Clipper permits expansion of the meaning of record.

- *Single-Order Bag:* An order bag that can contain only one order. The .ntx and .ndx files are examples of single-order bags.

- *Multiple-Order Bag:* An order bag that can contain any number of orders; a multiple-tag index. The .cdx and .mdx files are examples of multiple-order bags.

- *Maintainable Scoped Orders:* Scoped (filtered) orders created using the FOR clause. The FOR condition is stored in the index header. Orders of this type are correctly updated using the expression to reflect record updates, deletions and additions.
- *Non-Maintainable/Temporary Orders:* Orders created using the WHILE or NEXT clauses. These orders are useful because they can be created quickly. However, the conditions in these clauses are *not* stored in the index header. Therefore, orders of this type are *not* correctly updated to reflect record updates, deletions and additions. They are only for temporary use.
- *Lock List:* A list of the records that are currently locked in the work area.

Language Implementation

To support the RDD architecture and let you design applications that are independent of the data format you are using, many existing CA-Clipper commands and functions have been enhanced, and several new language elements have been added. The following tables summarize these changes and additions. See the *Reference Guide* for more detailed information on a particular item.

Enhanced Commands and Functions

Command/Function	Changes
APPEND FROM	VIA clause
COPY TO	VIA clause
DBAPPEND()	Terminology
GO	Terminology
DBSEEK	Terminology
INDEX	ALL, EVAL, EVERY, NEXT, RECORD, REST, TAG, and UNIQUE clauses
SEEK	SOFTSEEK option
SET INDEX	ADDITIVE clause
SET ORDER	IN, TAG clauses
DBSETINDEX()	Terminology
RECNO()	Terminology

New Commands and Functions

Command/Function	Description
BLOBDIRECTEXPORT()	Export the contents of a binary large object (BLOB) pointer to a file
BLOBDIRECTGET()	Retrieve data stored in a BLOB file without referencing a specific field
BLOBDIRECTIMPORT()	Import a file into a BLOB file and return a pointer to the data
BLOBDIRECTPUT()	Put data in a BLOB file without referencing a specific field
BLOBEXPORT()	Copy the contents of a BLOB, identified by its memo field number, to a file
BLOBGET()	Get the contents of a BLOB, identified by its memo field number
BLOBIMPORT()	Read the contents of a file as a BLOB, identified by a memo field number
BLOBROOTGET()	Retrieve the data from the root area of a BLOB file
BLOBROOTLOCK()	Obtain a lock on the root area of a BLOB file
BLOBROOTPUT()	Store data in the root area of a BLOB file
BLOBROOTUNLOCK()	Release the lock on a BLOB file's root area
DBFIELDINFO()	Return and optionally change information about a field
DBINFO()	Return and optionally change information about a database file opened in a work area
DBORDERINFO()	Return and optionally change information about orders and index files
DBRECORDINFO()	Return and optionally change information about a record
DBSEEK()	Move to the record having the specified key value
DELETE TAG	Delete a tag (order)
DBGOTO()	Position record pointer to a specific identity
DBRLOCK()	Lock the record at the current or specified identity
DBRLOCKLIST()	Return an array of the currently locked records
DBRUNLOCK	Release all or specified record locks
INDEX	Create an index file
MEMOSETSUPER()	Set the RDD inheritance chain for DBFMEMO

New Commands and Functions (cont.)

Command/Function	Description
ORDBAGEXT()	Return the order bag file extension
ORDBAGNAME()	Return the order bag name of a specific order
ORDCONDSET()	Set the condition and scope for an order
ORDCREATE()	Create an order in an order bag
ORDDESCEND()	Return and optionally change the descending flag of an order
ORDDESTROY()	Remove a specified order from an order bag
ORDISUNIQUE()	Return the status of the unique flag for a given order
ORDFOR()	Return the FOR expression of an order
ORDKEY()	Return the key expression of an order
ORDKEYADD()	Add a key to a custom-built order
ORDKEYCOUNT()	Return the number of keys in an order
ORDKEYDEL()	Delete a key from a custom-built order
ORDKEYGOTO()	Move to a record specified by its logical record number in the controlling order
ORDKEYNO()	Get the logical record number of the current record
ORDKEYVAL()	Get the key value of the current record from the controlling order
ORDLISTADD()	Add order bag contents or single order to the order list
ORDLISTCLEAR()	Clear the current order list
ORDLISTREBUILD()	Rebuild all orders in the order list of the current work area
ORDNAME()	Return the name of an order in the work area
ORDNUMBER()	Return the position of an order in the current order list
ORDSCOPE()	Set or clear the boundaries for scoping key values in the controlling order
ORDSETRELATION()	Relate a specified work area to the current work area
ORDSKIPUNIQUE()	Move the record pointer to the next or previous unique key in the controlling order
ORDSETFOCUS()	Set focus to an order in an order list
RDDLIST()	Return an array of the available Replaceable Database Drivers

New Commands and Functions (cont.)

Command/Function	Description
RDDNAME()	Return the name of the RDD active in the current or specified work area
RDDSETDEFAULT()	Set or return the default RDD for the application
SET DESCENDING	Change the descending flag of the controlling order
SET MEMOBLOCK	Change the block size for memo files
SET OPTIMIZE	Change the setting that determines whether to optimize using the open orders when processing a filtered database file
SET SCOPE	Change the top and/or bottom boundaries for scoping key values in the controlling order
SET SCOPEBOTTOM	Change the bottom boundary for scoping key values in the controlling order
SET SCOPETOP	Change the top boundary for scoping key values in the controlling order
SEEK	Search an order for a specified key value

User Interface Levels

We want to make it easy for you to quickly take advantage of the added functionality provided in CA-Clipper 5.3. In order to effectively use the RDDs, you should read the following discussions. They are provided as a means of identifying the degree of programming knowledge or CA-Clipper experience that will let you effectively use the RDD features.

For this purpose, the RDD feature set is divided into levels A and B, separating the simpler features from those that are more sophisticated. Tables listing the commands or functions that comprise these access levels are also supplied. In addition, an RDD Features Summary is provided in table form which outlines the features available in each driver. The commands and functions in both of these levels of access are described in the *Reference Guide, Volumes 1 and 2.*

Level A: Command-Level Interface

Level A. A simple command-level interface very similar to those found in other languages (e.g., dBASE IV, FoxPro). This is the primary access for new CA-Clipper users who may or may not be familiar with other languages.

The following table lists the commands and functions accessible by the CA-Clipper programmer with background in languages such as dBASE or FoxPro. The commands and functions in this table provide access to the additional features without requiring an advanced knowledge of CA-Clipper or other programming concepts.

Basic Commands and Functions

Command/Function	Changes
GOTO	Move the pointer to the specified identity
INDEX	Create an index file
SEEK	Search an order for a specified key value (see also, DBSEEK())
SET INDEX	Open one or more order bags in the current work area
SET ORDER	Select the controlling order
APPEND BLANK	Append a new record to the current lock list (see also, DBAPPEND())
RLOCK()	Lock the record at the current or specified identity (see also, DBRLOCK())
FLOCK()	Return an array of the current lock list (see also, DBRLOCKLIST())
UNLOCK	Release all or specified record locks (see also, DBRUNLOCK)

Level B: Function-Level Interface

Level B. CA-Clipper also adds a function level interface that not only allows access to the enhanced functionality of the drivers, but permits the building of higher-level functions using these composing behaviors. This level is meant for more experienced CA-Clipper users who need to take advantage of the full power of the driver and Order Management system.

The following table lists the DML and order management functions recommended to the intermediate or advanced CA-Clipper programmer. These functions provide the greatest flexibility in accessing the extended features of these drivers.

Advanced Commands and Functions

Command/Function	Description
BLOBDIRECTEXPORT()	Export the contents of a binary large object (BLOB) pointer to a file
BLOBDIRECTGET()	Retrieve data stored in a BLOB file without referencing a specific field
BLOBDIRECTIMPORT()	Import a file into a BLOB file and return a pointer to the data
BLOBDIRECTPUT()	Put data in a BLOB file without referencing a specific field
BLOBEXPORT()	Copy the contents of a BLOB, identified by its memo field number, to a file
BLOBGET()	Get the contents of a BLOB, identified by its memo field number
BLOBIMPORT()	Read the contents of a file as a BLOB, identified by a memo field number
BLOBROOTGET()	Retrieve the data from the root area of a BLOB file
BLOBROOTLOCK()	Obtain a lock on the root area of a BLOB file
BLOBROOTPUT()	Store data in the root area of a BLOB file
BLOBROOTUNLOCK()	Release the lock on a BLOB file's root area
DBAPPEND()	Append a new record to the current lock list
DBFIELDINFO()	Return and optionally change information about a field

Advanced Commands and Functions (cont.)

Command/Function	Description
DBINFO()	Return and optionally change information about a database file opened in a work area
DBORDERINFO()	Return and optionally change information about orders and index files
DBRECORDINFO()	Return and optionally change information about a record
DBRLOCK()	Lock the record at the current or specified identity
DBRLOCKLIST()	Return an array of the current lock list
DBRUNLOCK()	Release all or specified record locks
DELETE TAG	Delete a tag
MEMOSETSUPER()	Set the RDD inheritance chain for DBFMEMO
ORDBAGEXT()	Return the default order bag RDD extension
ORDBAGNAME()	Return the order bag name of a specific order
ORDCONDSET()	Set the condition and scope for an order
ORDCREATE()	Create an order in an order bag
ORDDESCEND()	Return and optionally change the descending flag of an order
ORDDESTROY()	Remove a specified order from an order bag
ORDFOR()	Return the FOR expression of an order
ORDISUNIQUE()	Return the status of the unique flag for a given order
ORDKEY()	Return the key expression of an order
ORDKEYADD()	Add a key to a custom-built order
ORDKEYCOUNT()	Return the number of keys in an order
ORDKEYDEL()	Delete a key from a custom-built order
ORDKEYGOTO()	Move to a record specified by its logical record number in the controlling order
ORDKEYNO()	Get the logical record number of the current record
ORDKEYVAL()	Get the key value of the current record from the controlling order
ORDLISTADD()	Add order bag contents or single order to the order list
ORDLISTCLEAR()	Clear the current order list

Advanced Commands and Functions (cont.)

Command/Function	**Description**
ORDLISTREBUILD()	Rebuild all orders in the order list of the current work area
ORDNAME()	Return the name of an order in the work area
ORDNUMBER()	Return the position of an order in the current order list
ORDSCOPE()	Set or clear the boundaries for scoping key values in the controlling order
ORDSETFOCUS()	Set focus to an order in an order list
ORDSETRELATION()	Relate a specified work area to the current work area
ORDSKIPUNIQUE()	Move the record pointer to the next or previous unique key in the controlling order
RDDLIST()	Return an array of the available Replaceable Database Drivers
RDDNAME()	Return the name of the RDD active in the current or specified work area
RDDSETDEFAULT()	Set or return the default RDD for the application
SET DESCENDING	Change the descending flag of the controlling order
SET MEMOBLOCK	Change the block size for memo files
SET OPTIMIZE	Change the setting that determines whether to optimize using the open orders when processing a filtered database file
SET SCOPE	Change the top and/or bottom boundaries for scoping key values in the controlling order
SET SCOPEBOTTOM	Change the bottom boundary for scoping key values in the controlling order
SET SCOPETOP	Change the top boundary for scoping key values in the controlling order

RDD Features

The following decision table summarizes the availability of key features across RDDs. It lists the features available in each RDD so you can use it as an aid in correct RDD implementation and data access.

RDD Features Summary

Item	NTX	NDX	MDX	CDX
Implicit record unlocking in single lock mode	Yes	Yes	Yes	Yes
Multiple record locks	Yes	Yes	Yes	Yes
Number of concurrent record locks	*1	*1	*1	*1
Order management (tag support)	Yes	Yes	Yes	Yes
Orders (tags) per order bag (file)	1	1	47	Unlimited
Number of order bags (files) per work area	15	15	15	Unlimited
Conditional indices (FOR clause)	Yes	No	Yes	Yes
Temporary (partial) indices (WHILE, ...)	Yes	No	No	Yes
Descending via DESCENDING clause	Yes	No	Yes	Yes
Unique via the UNIQUE clause	Yes	Yes	Yes	Yes
EVAL and EVERY clause support	Yes	No	No	Yes
Production/structural indices	No	No	Yes	Yes
Maximum key expression length (bytes)	256	256	220	255
Maximum FOR condition length (bytes)	256	N/A	261	255

*1 determined by available memory.

CA-Clipper 5.3 Order Management System

CA-Clipper includes an expanded Order Management system which provides a more effective and flexible way of indexing data. The main objective of the order management implementation is to raise the Xbase indexing paradigm from a low level of abstraction (Xbase database specific) to a higher, more robust, level. This higher level of abstraction allows the user to build new commands and functions.

Low-level abstraction refers to manipulation of discrete elements in the database architecture (i.e., field names and sizes, methods of handling controlling indices, etc.).

High-level abstraction refers to manipulation of general elements in a data source. It lets us, for example, set a controlling order without explicitly addressing the character of the data file structure. This higher level of abstraction was achieved by reviewing all the processes that indices have in common.

The order management function set was generically named (i.e., non-DBF specific) to provide a semantic that could encompass future RDD implementations that may not be file-bound. For example, an RDD could easily be created that orders (indexes) on a memory array, or other data structure, instead of a database. Therefore, all order management functions simply begin with "ORD" (for "order"). You will find the function names to be self-explanatory (e.g., ORDCREATE() creates an order, and ORDDESTROY() destroys an order).

Concept

An order is a set of keyed pairs that provides a logical ordering of the records in an associated database file. Each key in an order (index) is associated with a particular identity (record number) in the data set (database file). The records can be processed sequentially in key order, and any record can be located by performing a SEEK operation with the associated key value. An order never physically changes the data that it is applied against, but creates a different view of that data.

There are at least four basic types of processes that you can perform with an order:

1. *Ordering:* Changes the sequence in which you view the data records.
2. *Scoping:* Constrains the visibility of data to specified upper and lower bounds. Determines the range of data items included through a scoping rule, like the WHILE clause.

3. *Filtration:* Visibility of data is subject to conditional evaluation. Filtration determines which items of data are included through a filter rule, like the FOR clause.

4. *Translation:* Values in underlying data source are translated (or converted) in some form based on a selection criteria. For example:

   ```
   INDEX ON IIF(CUSTID > 1000, "NEW", "OLD")
   ```

The difference between scope and condition as it applies to FOR and WHILE is that the WHILE clause provides scope, but not filtering, but a FOR clause can provide both.

There are three primary elements in order management:

- *Order:* An order is a set that has two elements in it: an *order name,* which is a logical name that can be referenced, and an *order expression* which supplies the view of the data. The order name provides logical access to the expression and the order expression provides a way of viewing the underlying data source. Data ordering can also be modified to ascending or descending sequence.

  - *Order Name:* An order name is a symbolic name, like a file's alias, that you use to manipulate an order. The difference between an order name and the order number with which you would normally access indices (orders), is that the order name is stored in the index file. It is available each time you run the program, and is maintained by the system. The order number is generated each time the order is added to an order list and may change from one program execution to another. This makes order name the preferred means of referencing orders.

  - *Order Expression:* An order expression is any valid CA-Clipper expression. This is an index expression such as:

    ```
    CUSTLIST->CUSTID
    ```

    This expression produces the ordered view of the data. The values derived from this expression are sorted, and it is the relationship of these values to one another that provides the actual ordering.

- *Order Number:* An order number is provided by the order list. An order number is only valid as long as the work area to which it belongs is open.
  - Order numbers provide one of the services performed by order names, allowing you to access a specific order. In general, you should avoid accessing orders by number.
  - The ORDNUMBER() function returns the ordinal position of the specified <*orderName*> within the specified <*orderList*>.
- *Order Bag:* An order bag is an unsorted collection of orders. Each order contains two elements (order name and order expression). Each order bag *may* have *zero* to *n* orders. The maximum is determined by the RDD driver being used. Order bags are similar to multiple-index files in that there is no guarantee of any specific order within the container or *bag*. Within an order bag you can access specific orders by referencing a particular order name. Order bags have persistence between activations of the program.

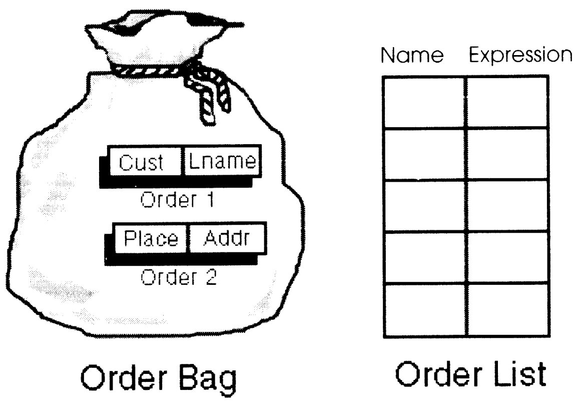


Figure 2-1: Order bag containing two orders before emptying to order list

- *Order List:* An order list *orders* the collection of orders that are associated with and active in the current work area. It provides access to the orders *active* within a given work area. Each work area has an order list, and there is only one order list per work area. An order list is created when a new work area is opened, and exists only as long as that work area is active. Once you close a work area, the order list ceases to exist.

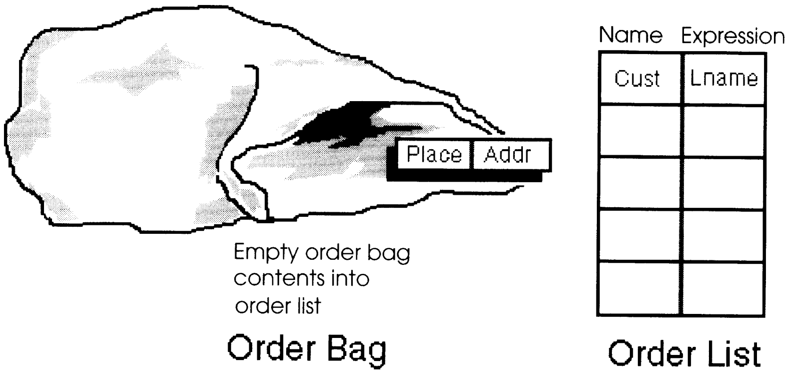


Figure 2-2: Order bag being emptied to order list

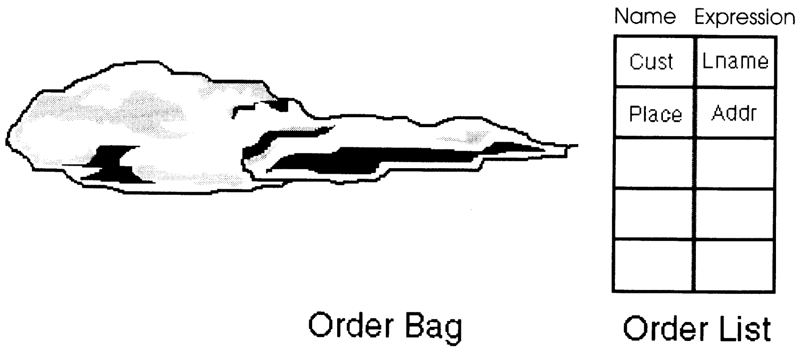
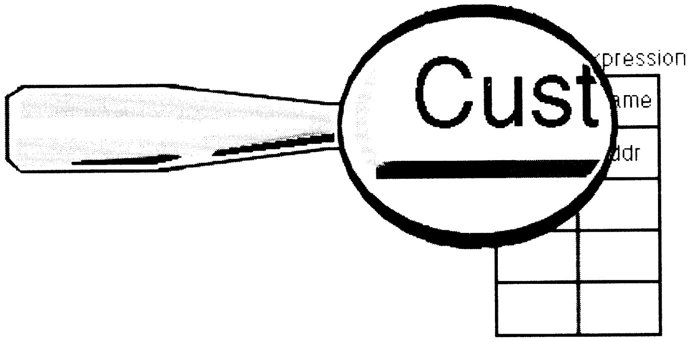


Figure 2-3: Order bag emptied to order list

When you SET INDEX TO, the contents of the order bag are emptied into the order list. At this point, the orders in the order list are active in the work area, where they will be updated as the data associated with the work area is modified. You may access an order in the list by its order number or by its order name. You should access an order by its name rather than a hard-coded ordinal position. You can make any order in the order list the controlling order by giving it focus, as explained below.

- *Order List Focus:* Order list focus is, essentially, a pointer to the order that is used to change the view of the data. It is synonymous with controlling order or controlling index, and defines the active index order. The SET ORDER TO command does not modify the order list in any way. It does not clear the active indices. It only changes the order list focus (the controlling order in the order list).



Order List Focus

Figure 2-4: SETting FOCUS to an order, "Name," in an order list

Notes

The following list contains specific information regarding order bag usage and limitations with DBFNDX and DBFNTX index files:

- *Single-order bags:* With DBFNDX and DBFNTX you can explicitly assign the order name within the order creation syntax. You can then use the order name in any command or function that accepts an order name (tag) as a parameter.
- *Single-order bag with INDEX ON:* Single-order bags may retain the order name between activations. During creation, DBFNTX stores an optionally supplied order name in the file's header for subsequent use. Therefore, the order name is *not* necessarily the same as that of the file. By contrast, DBFNDX cannot store an order name since this would prevent dBASE from accessing the file. By default DBFNDX orders inherit the name of their index file.

Summary

This chapter has introduced you to the RDD concept, giving you specific information on the architecture that implements RDDs in CA-Clipper. The basic terminology of RDDs has also been defined.

Finally, you have seen an overview of the language enhancements designed to make using RDDs straightforward and to let you build applications that do not depend on the RDD in use. The following chapters elaborate on these language enhancements, discussing syntax and usage in detail.

Chapter 3
DBFCDX Driver Installation and Usage

In This Chapter

DBFCDX is the FoxPro 2 compatible RDD for CA-Clipper. As such, it connects to the low-level database management subsystem in the CA-Clipper architecture. When you use the DBFCDX RDD, you add a number of new features including:

- FoxPro 2 file format compatibility
- Compact indices
- Compound indices
- Conditional indices
- Memo files smaller and more efficient than DBFNTX format
- An efficient storage mechanism for data which is not tabular in nature

This chapter explains how to install DBFCDX and how to use it in your applications. The following major topics are discussed:

- Overview of the DBFCDX RDD
- Installing DBFCDX Driver Files
- Linking the DBFCDX Driver
- Using the DBFCDX Driver

Overview of the DBFCDX RDD

The DBFCDX driver lets you create and maintain index/memo (.cdx/.fpt) files with features different from those supplied with the original DBFNTX driver and is compatible with files created under FoxPro 2. The new features are supplied in the form of several syntactical additions to the INDEX and REINDEX commands and some additional functions. Specifically, you can:

- Create indices smaller than those created with the DBFNTX driver. The key data is stored in a compressed format that substantially reduces the size of the index file.
- Create a compound index file that contains multiple indices (tags), making it possible to open several indices under one file handle. A single .cdx file may contain unlimited index keys in compressed format.
- Create conditional indices (FOR/WHILE/REST/NEXT).
- Create files with FoxPro 2 file format compatibility.
- Create memo files with a 4.2 GB file size limitation.
- Create memo files with a 1-byte minimum block size limitation.
- Recycle file space in memo files.
- Store any CA-Clipper data type (other than code blocks) in memo files.
- Use extensions to the CA-Clipper language (BLOB functions) for file and field management of memo files.

Compact Indices

Like FoxPro 2, the DBFCDX driver creates *compact indices*. This means that the key data is stored in a compressed format, resulting in a substantial size reduction in the index file. Compact indices store only the actual data for the index keys. Trailing blanks and duplicate bytes between keys are stored in one or two bytes. This allows considerable space savings in indices with much empty space and similar keys. Since the amount of compression is dependent on many variables, including the number of unique keys in an index, the exact amount of compression is impossible to predetermine.

Compound Indices

A *compound index* (.cdx) is an index file (called an order bag) that contains multiple indices (called tags). Compound indices make several indices available to your application while only using one file handle. Therefore, you can overcome the CA-Clipper index file limit of 15. The DBFCDX RDD permits opening as many indices per work area as there are available file handles and memory.

Once you open a compound index, all the tags contained in the file are automatically updated as the records are changed. A tag in a compound index is essentially identical to an individual index and supports all the same features. The first tag (in order of creation) in the compound index is, by default, the controlling index.

Note that you should avoid using SET ORDER TO [<*nOrder*>] with DBFCDX. Instead, use either:

```
SET ORDER TO  //Nullify order, leaving .cdx's open for updating
```

-OR-

```
SET ORDER TO TAG <cOrderName> [IN <xcOrderBagName>]
```

For more detailed information about the SET ORDER command, see the *Reference Guide, Volume* 2. Also, see the ORDSETFOCUS() function.

Conditional Indices

The DBFCDX driver can create indices with a built-in FOR clause. These are *conditional indices* in which the condition can be any expression, including a user-defined function. As the database is updated, only records that match the index condition are added to the index, and records that satisfied the condition before, but do not any longer, are automatically removed.

Expanded control over conditional indexing is supported with the revised INDEX and REINDEX command options as in the new DBFNTX driver.

Installing DBFCDX Driver Files

The DBFCDX driver is supplied as two files, DBFCDX.LIB and _DBFCDX.LIB.

The CA-Clipper installation program installs these files in the \CLIP53\LIB subdirectory on the drive that you specify, so you need not install the driver manually.

Linking the DBFCDX Database Driver

To link the DBFCDX database driver into an application program, you must specify DBFCDX.LIB to the linker in addition to your application object (.OBJ) files.

To link it, use:

```
EXOSPACE FI <appObjectList> LI DBFCDX, _DBFCDX
```

Note: These link commands all assume the LIB and OBJ environment variables are set to the standard locations. They also assume that the CA-Clipper programs were compiled without the /R option.

Using the DBFCDX Database Driver

To use FoxPro 2 files in a CA-Clipper program:

1. Place REQUEST DBFCDX at the beginning of your application or at the top of the first program (.prg) file that opens a database file using the DBFCDX driver.

2. Specify the VIA "DBFCDX" clause if you open the database file with the USE command.

   -OR-

3. Specify "DBFCDX" for the *<cDriver>* argument if you open the database file with the DBUSEAREA() function.

   -OR-

4. Use RDDSETDEFAULT("DBFCDX") to set the default driver to DBFCDX.

   Except in the case of REQUEST, the RDD name must be a literal character string or a variable. In all cases it is important that the driver name be spelled correctly using uppercase letters.

The following program fragments illustrate:

```
REQUEST DBFCDX
.
.
.
USE Customers INDEX Name, Address NEW VIA "DBFCDX"
```

-OR-

```
REQUEST DBFCDX
RDDSETDEFAULT( "DBFCDX" )
.
.
.
USE Customers INDEX Name, Address NEW
```

Using .cdx/.fpt and .ntx/.dbt Files Concurrently

You can use both .cdx/.fpt and .ntx/.dbt files concurrently in a CA-Clipper program like this:

```
// (.ntx) file using default DBFNTX driver
USE File1 INDEX File1 NEW

// (.cdx) files using DBFCDX driver (with non-production .cdx
// files)
USE File2 VIA "DBFCDX" INDEX File2 NEW
```

Note, however, that you cannot use .cdx and .ntx files in the same work area. For example, the following does not work:

```
USE File1 VIA "DBFNTX" INDEX File1.ntx, File2.cdx
```

You can use .cdx and .idx files concurrently in the same work area, but this is not recommended since a .cdx file may additionally contain the desired index key. However, you can use the .fpt format with the .ntx index. You must use the DBFMEMO RDD, and *not* the DBFCDX RDD, to achieve this combination.

File Maintenance Under DBFCDX

When an existing tag in a compound index (.cdx) is rebuilt using INDEX ON...TAG..., the space used by the original tag is not automatically reclaimed. Instead, the new tag is added to the end of the file, increasing file size. Because of this, you should use INDEX...ON...TO to create temporary files and not create them as tags in the production .cdx.

You can use the REINDEX command to "pack" the index file. REINDEX rebuilds each tag, eliminating any unused space in the file.

If you rebuild your indices on a regular basis, you should either delete your .cdx files before rebuilding the tags or use the REINDEX command to eliminate wasted space after recreating a tag within the .cdx file.

The .fpt files do not require maintenance under normal conditions. They will not grow more than 10 to 20 percent larger than their packed size even on a network with a lot of file activity. However, if you want to eliminate this working slack space from the .fpt files (for example, when shipping data to a customer), use the COPY TO command to create a .fpt file that is as small as possible. Note that the PACK command does not eliminate the extra space that may be present. However, the space is available for reuse.

DBFCDX and Memo Files

The DBFCDX driver uses FoxPro-compatible memo (.fpt) files to store data for memo fields. These memo files have a default block size of 32 bytes rather than the 512-byte default for .dbt files. In addition, the block size can be set as low as 1 byte which yields a true variable length field memo file. However, using a block size less than 32 bytes will render the file incompatible with FoxPro.

DBFCDX memo files can store any type of data except code blocks and objects. In addition, they can store data larger than 64K (remember that strings in CA-Clipper variables are still limited to 64K). While .dbt files use an end of file marker (ASCII 26) at the end of a memo entry, .fpt files store the length of the entry. This not only eliminates the problems normally encountered with storing binary data in a memo field, but also speeds up memo field access since the data need not be scanned to determine the length.

Note that FoxPro is only capable of reading and writing character data to a memo file. Writing arrays and other data types to a memo file is a CA-Clipper extension to the .fpt file format, and FoxPro will not be able to read these data types. In addition, FoxPro is not able to read character data greater than 64K.

Tips for Using DBFCDX

Below are several tips for using DBFCDX:

1. Make sure index extensions are not hard-coded in your application. The default extension for DBFCDX indices is .cdx, *not* .ntx. You can, however, still use .ntx as the extension as long as you specify the extension when you create your indices. The best way to determine index extensions in an application is to call ORDBAGEXT().

   For example, if you currently use the following code to determine the existence of an index file,

   ```
   IF .NOT. FILE("Index.ntx")
      INDEX ON field TO index
   ENDIF
   ```

   change the code to include the ORDBAGEXT() function, as follows:

   ```
   IF .NOT. FILE("index"+ORDBAGEXT())
      INDEX ON field TO index
   ENDIF
   ```

   Caution! *DBFCDX supports both .cdx and .idx files; and ORDBAGEXT() returns the default index extension of the driver loaded, not the actual index file extension.*

2. If your application uses memo fields, you should convert your .dbt files to .fpt files.

   There are some good reasons for using .fpt files. Most important is the 32-byte block size (or 1-byte blocks, which are even more efficient but are not compatible with most implementations of .fpt files). CA-Clipper's .dbt files use a fixed block size of 512 bytes, which means that every time you store even 1 byte in a memo field CA-Clipper uses 512 bytes to store it. If the data in a memo field grows to 513 bytes, then *two* blocks are required.

   When creating .fpt files, the block size is set at 32 bytes which is the optimal size while retaining .fpt compatibility. A simple conversion from .dbt files to .fpt files will generally shrink your memo files by approximately 30 percent (50 percent if you set the block size to 1).

3. Add DBFCDX.LIB and _DBFCDX.LIB as libraries to your link command or link script.
4. If you receive an error message indicating that there is a problem with the .cdx file when creating a production .cdx index using

   ```
   INDEX ON <expKey> TAG <OrderName>
   ```

   you may reset the .dbf header. Specifically, the character at OffSet 1Ch (16 + 12) needs to be changed or reset to 0.

   To reset the .dbf header for a missing or corrupt production .cdx file, use the following low-level utility:

   ```
   FUNCTION ResetDBHdr( cFileName )

      LOCAL nHandle, cBuffer, nOffSet := 28

      cBuffer := CHR(0)

      nHandle := FOPEN( cFileName, FO_READWRITE )
         IF !( nHandle == -1 )
            FSEEK( nHandle, nOffSet )
            FWRITE( nHandle, cBuffer )
            FCLOSE( nHandle )
            ? "All done"
         ELSE
            ? "Error opening the file.  DOS error: ", FERROR()
         ENDIF

      RETURN ( NIL )
   ```

Summary

In this chapter, you were given an overview of the features and benefits of the DBFCDX RDD. You also learned how to link this driver and how to use it in your applications.

Chapter 4
DBFMDX Driver Installation and Usage

In This Chapter

DBFMDX is the dBASE IV compatible RDD for CA-Clipper. This driver provides .dbf, .dbt, and .mdx file format compatibility.

This chapter explains how to install DBFMDX and how to use it in your applications. The following major topics are discussed:

- Overview of the DBFMDX RDD
- Installing DBFMDX Driver Files
- Linking the DBFMDX Driver
- Using the DBFMDX Driver

Overview of the DBFMDX RDD

The DBFMDX database driver provides dBASE IV compatibility, including access to .dbf, .mdx, and .dbt file formats. The driver also supports dBASE IV compatible file and record locking schemes, allowing shared access between CA-Clipper and dBASE IV programs.

Note: CA-Clipper does not currently provide dBASE V compatibility.

Installing DBFMDX Driver Files

The DBFMDX database driver is supplied as the file, DBFMDX.LIB.

The CA-Clipper installation program installs this driver in the \CLIP53\LIB subdirectory on the drive that you specify, so you need not install the driver manually.

Linking the DBFMDX Database Driver

To link the DBFMDX database driver into an application program, you must specify DBFMDX.LIB to the linker in addition to your application object (.OBJ) files.

To link it, use:

```
EXOSPACE FI <appObjectList> LI DBFMDX
```

Note: These link commands all assume the LIB and OBJ environment variables are set to the standard locations. They also assume that the CA-Clipper programs were compiled without the /R option.

Using the DBFMDX Database Driver

To use .mdx files in a CA-Clipper program:

1. Place REQUEST DBFMDX at the beginning of your application or at top of the first program (.prg) file that opens a database file using the DBFMDX driver.

2. Specify the VIA "DBFMDX" clause if you open the database file with the USE command.

   -OR-

3. Specify "DBFMDX" for the <*cDriver*> argument if you open the database file with the DBUSEAREA() function.

   -OR-

4. Use RDDSETDEFAULT("DBFMDX") to set the default driver to DBFMDX.

Except in the case of REQUEST, the RDD name must be a literal character string or a variable. In all cases it is important that the driver name be spelled correctly using uppercase letters.

Tips for Using DBFMDX

1. Though the DBFMDX driver supports numbered indices, you should avoid using them (e.g., in operations like SET ORDER and INDEXKEY()). In a multiple-index system you do not have the absolute control of the numeric position of an open index that you have in a single-index system. As you add, delete, and rebuild index tags, their numeric position may change. Therefore, you should make all command references by name and avoid using SET ORDER TO [<*nOrder*>] with DBFMDX. Instead, use either:

   ```
   SET ORDER TO  //Nullify order, leaving .mdx's open for updating
   ```

   -OR-

   ```
   SET ORDER TO TAG <cOrderName> [IN <xcOrderBagName>]
   ```

   For more detailed information about the SET ORDER command, see the *Reference Guide, Volume 2*. Also, see the ORDSETFOCUS() function.

2. When creating a production .mdx index using the following,

   ```
   INDEX ON <expKey> TAG <OrderName>
   ```

   you may receive an error message indicating that the .dbf file header is corrupted. Specifically, the character at OffSet 1Ch (16 + 12) needs to be changed or reset to 0.

   To reset the .dbf header for a missing or corrupt production .mdx file, use the following low-level utility:

   ```
   FUNCTION ResetDBHdr( cFileName )

      LOCAL nHandle, cBuffer, nOffSet := 28

      cBuffer := CHR(0)

      n Handle := FOPEN( cFileName, FO_READWRITE )
         IF ! ( nHandle == -1 )
            FSEEK( nHandle, nOffSet )
            FWRITE( nHandle, cBuffer )
            FCLOSE( nHandle )
            ? "All done"
         ELSE
            ? "Error opening the file.  DOS error: ", FERROR()
         ENDIF

      RETURN ( NIL )
   ```

Summary

In this chapter, you were given an overview of the features and benefits of the DBFMDX RDD. You also learned how to link this driver and how to use it in your applications.

Chapter 5
DBFNDX Driver Installation and Usage

In This Chapter

DBFNDX is the dBASE III PLUS compatible RDD for CA-Clipper. The DBFNDX driver uses the CA-Clipper driver architecture to access dBASE III PLUS compatible index files within a CA-Clipper program.

This chapter explains how to install DBFNDX and how to use it in your applications. The following major topics are discussed:

- Overview of the DBFNDX RDD
- Installing DBFNDX Driver Files
- Linking the DBFNDX Driver
- Using the DBFNDX Driver
- Compatibility with dBASE III PLUS

Overview of the DBFNDX RDD

The DBFNDX database driver allows creation, access, and updating of dBASE III and dBASE III PLUS compatible index (.ndx) files. Index (.ndx) files created with CA-Clipper are exactly the same as those created by dBASE III PLUS. All operations that can be performed on standard CA-Clipper index (.ntx) files can be performed on .ndx files using the DBFNDX database driver.

In a network environment, the DBFNDX driver supports the CA-Clipper file and record locking scheme. The multi-user behavior is the same as the default DBFNTX driver. This means that the DBFNDX database driver supports concurrent access to .ndx files between CA-Clipper applications only. Concurrent access to .ndx files between dBASE III PLUS and CA-Clipper programs is *not* supported.

Important! *Updating database (.dbf) and index (.ndx) files shared between dBASE III PLUS and CA-Clipper programs may corrupt the .dbf file and any of its associated .ndx files.*

Installing DBFNDX Driver Files

The DBFNDX database driver is supplied as the file, DBFNDX.LIB.

The CA-Clipper installation program installs this driver in the \CLIP53\LIB subdirectory on the drive that you specify, so you need not install the driver manually.

Linking the DBFNDX Database Driver

To link the DBFNDX database driver into an application program, you must specify DBFNDX.LIB to the linker in addition to your application object (.OBJ) files.

To link it, use:

```
EXOSPACE FI <appObjectList> LI DBFNDX
```

Note: These link commands all assume that the LIB and OBJ environment variables are set to the standard locations. They also assume that the CA-Clipper programs were compiled without the /R option.

Using the DBFNDX Database Driver

To use .ndx files in a CA-Clipper program:

1. Place a REQUEST DBFNDX at the beginning of your application or at the top of the first program (.prg) file that opens a database file using the DBFNDX driver.

2. Specify the VIA "DBFNDX" clause if you open the database file with the USE command.

   -OR-

3. Specify "DBFNDX" for the *<cDriver>* argument if you open the database file with the DBUSEAREA() function.

   -OR-

4. Use RDDSETDEFAULT("DBFNDX") to set the default driver to DBFNDX.

   Except in the case of REQUEST, the RDD name must be a literal character string or a variable. In all cases it is important that the driver name be spelled correctly using uppercase letters.

The following program fragments illustrate:

```
REQUEST DBFNDX
.
.
.
USE Customers INDEX Name, Address NEW VIA "DBFNDX"
```

-OR-

```
REQUEST DBFNDX
RDDSETDEFAULT( "DBFNDX" )
.
.
.
USE Customers INDEX Name, Address NEW
```

Using .ntx and .ndx Files Concurrently

You can use .ndx and .ntx files concurrently in a CA-Clipper program like this:

```
REQUEST DBFNDX

// (.ntx) file using default DBFNTX driver
USE File1 INDEX File1 NEW

// (.ndx) files using DBFNDX driver
USE File2 VIA "DBFNDX" INDEX File2 NEW
```

Note, however, that you *cannot* use .ndx and .ntx files in the same work area. For example, the following does not work:

```
USE File1 VIA "DBFNDX" INDEX File1.ntx, File2.ndx
```

Compatibility with dBASE III PLUS

When accessing dBASE III PLUS (.ndx) files, there are several compatibility issues of which you must be aware. These issues are discussed below.

Supported Data Types

The DBFNDX database driver supports the following data types for key expressions:

- Character
- Numeric
- Date (using the DTOS() function, *not* DTOC())

This is consistent with dBASE III PLUS.

The DBFNDX database driver does not support indexing with logical key expressions as does the default DBFNTX database driver. This is actually a dBASE III PLUS limitation and is not supported by the DBFNDX driver in order to enforce compatibility with dBASE III PLUS.

To work around this limitation, index logical values by converting them to character values like this:

```
INDEX ON IIF(<lExp>, "T", "F") TO <logicalIndex>
```

Supported Key Expressions

When you create .ndx files using the DBFNDX driver, you must use only CA-Clipper or user-defined functions compatible with dBASE III PLUS. Use of the other functions will render the .ndx file unreadable in dBASE III PLUS.

FIND vs. SEEK

In CA-Clipper, you can use the FIND compatibility command only to locate keys in indices where the index key expression is character type. This differs from dBASE III PLUS where FIND supports character and numeric key values.

Note: In CA-Clipper programs, always use the SEEK command or the DBSEEK() function to search an index for a key value.

The DBFNDX driver lets you recover from a data type error raised during a FIND or SEEK. However, since Error:canDefault, Error:canRetry, and Error:canSubstitute are set to false (.F.), you should use BEGIN SEQUENCE...END to handle a SEEK or FIND data type error. Within the error block for the current operation, issue a BREAK() using the error object the DBFNDX database driver generates, like this:

```
bOld := ERRORBLOCK({|oError| BREAK(oError)})
.
.
.
BEGIN SEQUENCE
   SEEK xVar
RECOVER USING oError
// Recovery code
END
.
.
.
ERRORBLOCK(bOld)
```

There is an extensive discussion of the effective use of the CA-Clipper error system in the "Error Handling Strategies" chapter of the *Programming and Utilities Guide.*

Sharing Data on a Network

As mentioned above, the DBFNDX driver does not support dBASE III PLUS file and record locking schemes. Instead, the DBFNDX driver supports the DBFNTX file and record locking scheme. This means that if the same database and index files are open in CA-Clipper and dBASE III PLUS, CA-Clipper program locks are not visible to dBASE III PLUS and vice versa.

Warning! *Database integrity is not guaranteed and index corruption will occur if CA-Clipper and dBASE III PLUS programs attempt to write to a database or index file at the same time. For this reason, concurrent use of the same database (.dbf) and index (.ndx) files by dBASE III PLUS and CA-Clipper programs is strongly discouraged and not supported by Computer Associates.*

Compatibility with dBASE IV

Specific compatibility with dBASE IV is provided through the DBFMDX driver. It includes .dbf, .mdx, and .dbt file format compatibility and is described in detail in the previous chapter.

Summary

In this chapter, you were given an overview of the features and benefits of the DBFNDX RDD. You learned how to link this driver and how to use it in your applications. In addition you were given an overview of the compatibility issues.

Chapter 6
DBFNTX Driver Installation and Usage

In This Chapter

DBFNTX is the default RDD for CA-Clipper. This new database driver replaces the DBFNTX database driver supplied with earlier versions of CA-Clipper and adds a number of new indexing features. With DBFNTX, you can:

- Create conditional indices by specifying a FOR condition
- Create indices using a record scope or WHILE condition, allowing you to INDEX based on the order of another index
- Create both ascending and descending order indices
- Specify an expression that is evaluated periodically during indexing in order to display an index progress indicator

This chapter explains how to install DBFNTX and how to use it in your applications. The following major topics are discussed:

- Overview of the DBFNTX RDD
- Installing DBFNTX Driver Files
- Linking the DBFNTX Driver
- Using the DBFNTX Driver
- Compatibility with dBASE III PLUS

Overview of the DBFNTX RDD

As an update of the default database driver, DBFNTX is linked into and used automatically by your application unless you compile using the /R option.

New Features

The replaceable driver lets you create and maintain index (.ntx) files using features above and beyond those supplied with the previous DBFNTX driver. The new indexing features are supplied in the form of several syntactical additions to the INDEX and REINDEX commands. Specifically you can:

- Specify full record scoping and conditional filtering using the standard ALL, FOR, WHILE, NEXT, REST, and RECORD clauses
- Create an index while another controlling index is still active
- Monitor indexing as each record (or a specified record number interval) is processed using the EVAL and EVERY clauses
- Eliminate separate coding for descending order keys using the DESCENDING clause

Compatibility

Index (.ntx) files created with the original DBFNTX driver are compatible with DBFNTX and can be used in new applications without reindexing. Index (.ntx) files created with this version of DBFNTX will also work with previous CA-Clipper applications provided that you use no FOR, WHILE, <*scope*>, or DESCENDING clauses.

Important! *Indices produced with DBFNTX using FOR or DESCENDING are incompatible with earlier version .ntx files. If you attempt to access them with the original DBFNTX database driver or programs compiled with versions earlier than CA-Clipper 5.2, you will get an unrecoverable runtime error. In CA-Clipper, this generates an "index corrupted" error message, causing the application to terminate.*

Installing DBFNTX Driver Files

The DBFNTX driver is supplied as the file, DBFNTX.LIB.

The CA-Clipper installation program installs this driver as the default in the \CLIP53\LIB subdirectory on the drive that you specify, so you need not install the driver manually.

Linking the DBFNTX Database Driver

Since DBFNTX is the default database driver for CA-Clipper, there are no special instructions for linking. Unless you specify the /R option when you compile, the new driver will be linked into each program automatically if you specify a USE command or DBUSEAREA() function without an explicit request for another database driver. The driver is also linked if you specify an INDEX or REINDEX command with any of the new features.

Using the DBFNTX Database Driver

In applications written for the new DBFNTX driver, you can use the INDEX and REINDEX commands exactly as you have used them in the past. The index (.ntx) files you create and maintain in this way are completely compatible with those created using previous versions of the driver.

Changes to existing code are necessary only if you use the new indexing features. The .ntx files you create using the new features will have a slightly different header file and cannot be used by programs linked with a previous version of the driver.

Using .ntx and .ndx Files Concurrently

You can use .ntx and .ndx files concurrently in a CA-Clipper program like this:

```
// (.ntx) file using default DBFNTX driver
USE File1 INDEX File1 NEW

// (.ndx) files using DBFNDX driver
USE File2 VIA "DBFNDX" INDEX File2 NEW
```

Note, however, that you *cannot* use .ntx and .ndx files in the same work area. For example, the following does not work:

```
USE File1 VIA "DBFNDX" INDEX File1.ntx, File2.ndx
```

Compatibility with dBASE III PLUS

The default DBFNTX driver makes CA-Clipper programs behave differently from traditional dBASE programs. Some of these differences are discussed below:

Supported Data Types

The DBFNTX database driver supports the following dBASE III PLUS compatible data types for key expressions:

- Character
- Numeric
- Date
- Logical

Supported Key Expressions

When you create .ntx files using the DBFNTX driver, you can use all CA-Clipper or user-defined functions compatible with dBASE III PLUS, as well as other functions accepted by the extended CA-Clipper functionality.

Error Handling

The indexing behavior of DBFNTX and DBFNDX in a CA-Clipper application is identical unless otherwise noted. With the default DBFNTX driver, you can handle most errors using BEGIN SEQUENCE...END as illustrated in the next section.

FIND vs. SEEK

In CA-Clipper, you can use the FIND command only to locate keys in indices where the index key expression is a character data type. This differs from dBASE III PLUS where FIND supports character and numeric key values.

Note: In CA-Clipper programs, always use the SEEK command or the DBSEEK() function to search an index for a key value.

The DBFNTX driver lets you recover from data type errors raised during a FIND or SEEK. However, since Error:canDefault, Error:canRetry and Error:canSubstitute are set to false (.F.), you should use BEGIN SEQUENCE...END to handle such SEEK or FIND data type errors. Within the error block for the current operation, issue a BREAK() using the error object that the DBFNTX database driver generates, like this:

```
bOld := ERRORBLOCK({|oError| BREAK(oError)})
.
.
.
BEGIN SEQUENCE
   SEEK xVar
RECOVER USING oError
   // Recovery code
END
.
.
.
ERRORBLOCK(bOld)
```

There is an extensive discussion of the effective use of the CA-Clipper error system in the "Error Handling Strategies" chapter of the *Programming and Utilities Guide.*

Sharing Data on a Network

The DBFNTX driver provides file and record locking schemes that are different from dBASE III PLUS schemes. This means that if the same database and index files are open in CA-Clipper and in dBASE III PLUS, CA-Clipper program locks are not visible to dBASE III PLUS and vice versa.

Warning! *Database integrity is not guaranteed and index corruption will occur if CA-Clipper and dBASE III PLUS programs attempt to write to a database or index file at the same time. Therefore, concurrent use of the same database (.dbf) and index (.ndx) files by dBASE III PLUS and CA-Clipper programs is strongly discouraged and not supported by Computer Associates.*

Summary

In this chapter, you were given an overview of the new features of the default DBFNTX RDD. You learned how this driver is automatically linked and how to use it in your applications, and were given an overview of the compatibility issues.

Chapter 7
DBFMEMO Driver Installation and Usage

In This Chapter

DBFMEMO provides very efficient and flexible memo file features which can be used in conjunction with DBFNTX, DBFMDX, or other third party index/database drivers. As such, it inherits the index and database characteristics from one of these other drivers while maintaining its own memo file storage characteristics. When you use the DBFMEMO RDD, you add a number of new features including:

- All non-memo features of the "super" driver (DBFNTX, DBFMDX, etc.)
- An efficient storage mechanism for data which is not tabular in nature

This chapter explains how to install DBFMEMO and how to use it in your applications. The following major topics are discussed:

- Overview of the DBFMEMO RDD
- Installing DBFMEMO Driver Files
- Linking the DBFMEMO Driver
- Using the DBFMEMO Driver

Overview of the DBFMEMO RDD

The DBFMEMO driver lets you create and maintain index files compatible with one driver (e.g., DBFNTX) and, simultaneously, create and maintain memo (.dbv or .fpt) files with features compatible with or even more powerful than .fpt files created under FoxPro 2.

As the DBFMEMO RDD can only maintain memo data, it must be used in conjunction with another driver capable of handling index and .dbf functionality. The features this driver will provide (in addition to those it inherits from its "super" driver) are listed below:

- A 4.2 GB file size limitation
- A 1-byte minimum block size limitation
- An efficient technique for recycling file space
- The capability to store any CA-Clipper data type (other than code blocks)
- Extensions to the CA-Clipper language (BLOB functions) for file and field management

Installing DBFMEMO Driver Files

The DBFMEMO driver is supplied as the file, DBFMEMO.LIB.

The CA-Clipper installation program installs this driver in the \CLIP53\LIB subdirectory on the drive that you specify, so you need not install the driver manually.

Linking the DBFMEMO Database Driver

To link the DBFMEMO database driver into an application program, you must specify DBFMEMO.LIB to the linker, in addition to your application object (.OBJ) files and the "super" driver (in this example, DBFNTX is used as the "super" driver).

To link with the protected mode linker, use:

```
EXOSPACE FI <appObjectList> LI DBFMEMO, [<superRDDLib>]
```

Note: These link commands all assume the LIB and OBJ environment variables are set to the standard locations. They also assume that the CA-Clipper programs were compiled without the /R option. When using the DBFNTX driver as the "super" to DBFMEMO, it is linked by default. Specify other database drivers to be used as the "super" where <*superRDDLib*> is stated as optional in the link command.

Using the DBFMEMO Database Driver

The DBFMEMO driver must be used in conjunction with a "super" driver. In the following discussion, we will assume DBFNTX to be the chosen "super" driver. However, we could have chosen the DBFMDX or DBFNDX drivers and obtained similar results:

1. Place REQUEST DBFMEMO at the beginning of your application or at the top of the first program (.prg) file that opens a database file using the DBFMEMO driver.
2. Call MEMOSETSUPER("DBFNTX") at the beginning of your application, or at the very least, before you open any database files.
3. Specify the VIA "DBFMEMO" clause if you open the database file with the USE command.

   -OR-

4. Specify "DBFMEMO" for the <*cDriver*> argument if you open the database file with the DBUSEAREA() function.

   -OR-

5. Use RDDSETDEFAULT("DBFMEMO") to set the default driver to DBFMEMO.

   Except in the case of REQUEST, the RDD name must be a literal character string or a variable. In all cases it is important that the driver name be spelled correctly using uppercase letters.

The following program fragments illustrate:

```
REQUEST DBFMEMO
REQUEST DBFNTX
MEMOSETSUPER("DBFNTX")
.
.
USE Customers INDEX Name, Address NEW VIA "DBFMEMO"
```

-OR-

```
REQUEST DBFMEMO
REQUEST DBFNTX
RDDSETDEFAULT( "DBFMEMO" )
MEMOSETSUPER( "DBFNTX" )
.
.
.
USE Customers INDEX Name, Address NEW
```

Note: The *REQUEST DBFNTX* compiler declaration is the default in CA-Clipper. When stating other drivers as "super" to DBFMEMO, place the REQUEST statement where DBFNTX is shown.

File Maintenance Under DBFMEMO

For index file maintenance under DBFMEMO, refer to the similar section of the chosen super driver.

The .dbv files do not require maintenance under normal conditions. They will not grow more than 10 to 20 percent larger than their packed size even on a network with a lot of file activity. However, if you want to eliminate this working slack space from the .dbv files (for example, when shipping data to a customer), use the COPY TO command to create a .dbv file that is as small as possible. Note that the PACK command does not eliminate the slack space that may be present; however, the slack space made available by a PACK is available for reuse.

DBFMEMO and Memo Files

The DBFMEMO driver, by default, uses a non-FoxPro-compatible memo (.dbv) file to store data for memo fields. These memo files have a default block size of 1 byte rather than the 512-byte default for .dbt files or the 32 bytes of the FoxPro 2 .fpt file. The block size can be set to 32 bytes to achieve memo compatibility with FoxPro; however, index compatibility will depend on your chosen super driver.

DBFMEMO memo files can store any type of data except code blocks. In addition, they can store data larger than 64 K (remember that Clipper string variables are still limited to 64 K). While .dbt files use an end of file marker (ASCII 26) at the end of a memo entry, .dbv files store the length of the entry. This not only eliminates the problems normally encountered with storing binary data in a memo field, but also speeds up memo field access since the data need not be scanned to determine the length.

Note that FoxPro is only capable of reading and writing character data to a memo file and that the block size of the memo file must be equal to or larger than 32. Writing arrays and other data types to a memo file is a CA-Clipper extension to the .dbv file format, and FoxPro will not be able to read these data type. In addition, FoxPro is not able to read character data greater than 64 K.

Tips for Using DBFMEMO

Below are some tips for using DBFMEMO:

1. If your application uses memo fields, you should convert your .dbt files to .dbv files.

   There are some good reasons for using .dbv files. The CA-Clipper .dbt files use a fixed block size of 512 bytes, which means that every time you store even 1 byte in a memo field CA-Clipper uses 512 bytes to store it. If the data in a memo field grows to 513 bytes, then *two* blocks are required. The .dbv file, on the other hand, requires only the exact space of the data being stored.

   When creating .dbv files, the block size is set at 1 byte which is the optimal size, although it is not .fpt compatible. A simple conversion from .dbt files to .dbv files will generally shrink your memo files by approximately 50 percent.

2. Add DBFMEMO.LIB as a library to your link command or link script.

Summary

In this chapter, you were given an overview of the features and benefits of the DBFMEMO RDD. You also learned how to link this driver and how to use it in your applications.

Chapter 8
DBFBLOB Driver Installation and Usage

In This Chapter

DBFBLOB is a special RDD for CA-Clipper. When you use the DBFBLOB RDD, you have access to a number of new features including:

- Stand-alone "blob" (.dbv) files that do not require associated database (.dbf) files
- An efficient storage mechanism for data which is not tabular in nature

This chapter explains how to install DBFBLOB and how to use it in your applications. The following major topics are discussed:

- Overview of the DBFBLOB RDD
- Installing DBFBLOB Driver Files
- Linking the DBFBLOB Driver
- Using the DBFBLOB Driver

Overview of the DBFBLOB RDD

The DBFBLOB RDD is designed to give you an alternative mechanism for storing and retrieving memo fields and to give you direct control over managing the file used to store the data (called a BLOB file, for Binary Large OBject).

The DBFBLOB driver features:

- A 4.2 GB file size limitation
- A 1-byte minimum block size limitation
- An efficient technique for recycling file space
- The capability to store any CA-Clipper data type (other than code blocks)
- Extensions to the CA-Clipper language (BLOB functions) for file and field management

The DBFBLOB RDD lets you create and maintain a memo (.dbv) file without requiring an associated database (.dbf) file. This lets you create one file that would contain what you might normally put in several memo (.dbt) files. It is especially useful for storing startup information such as user IDs, customizable options such as color settings, etc.

As there is no associated .dbf file with the DBFBLOB RDD, there is an added level of management that is required by the programmer. Under a conventional DBF RDD, memo field data is stored in the memo file and a *reference value* to that data is automatically stored in the 10-byte memo field in the .dbf file.

The DBFBLOB RDD uses functions such as BLOBDIRECTPUT() to store data in the memo file and returns the reference value to your program instead of storing it automatically in a .dbf memo field record pointer. The programmer then assumes the responsibility of saving the *reference value,* which if lost is equivalent to losing the data.

The function BLOBROOTPUT() serves as the solution to this problem. BLOBROOTPUT() will store one (and only one) piece of data which can be retrieved with BLOBROOTGET(). If your program only requires one piece of data to be stored independently (such as an array of user settings), simply use BLOBROOTPUT() to store the data.

However, if the program requires that many pieces of data need to be stored independently, you can make multiple calls to BLOBDIRECTPUT() and store the returned reference values into an array. Then, store the *array* of reference values in the root of the memo file with BLOBROOTPUT().

Installing DBFBLOB Driver Files

The DBFBLOB driver is supplied as the file DBFBLOB.LIB.

The CA-Clipper installation program installs this driver in the \CLIP53\LIB subdirectory on the drive that you specify, so you need not install the driver manually.

Linking the DBFBLOB Database Driver

To link the DBFBLOB database driver into an application program, you must specify DBFBLOB.LIB to the linker in addition to your application object (.OBJ) files.

To link with the protected mode linker, use:

```
EXOSPACE FI <appObjectList> LI DBFBLOB
```

Note: These link commands all assume the LIB and OBJ environment variables are set to the standard locations. They also assume that the CA-Clipper programs were compiled without the /R option.

Using the DBFBLOB Database Driver

To use the DBFBLOB RDD in a CA-Clipper program:

1. Place REQUEST DBFBLOB at the beginning of your application or at the top of the first program (.prg) file that opens a database file using the DBFBLOB driver.

2. Specify the VIA "DBFBLOB" clause if you open the database file with the USE command.

   -OR-

3. Specify "DBFBLOB" for the *<cDriver>* argument if you open the database file with the DBUSEAREA() function.

   -OR-

4. Use RDDSETDEFAULT("DBFBLOB") to set the default driver to DBFBLOB.

   Except in the case of REQUEST, the RDD name must be a literal character string or a variable. In all cases it is important that the driver name be spelled correctly using uppercase letters.

The following program fragments illustrate:

```
REQUEST DBFBLOB
.
.
.

// Opening Customers.Dbv
USE Customers NEW VIA "DBFBLOB"
```

-OR-

```
REQUEST DBFBLOB
RDDSETDEFAULT( "DBFBLOB" )
.
.
.
// Opening Customers.Dbv
USE Customers NEW
```

Summary

In this chapter, you were given an overview of the features and benefits of the DBFBLOB RDD. You also learned how to link this driver and how to use it in your applications.

Chapter 9
Alternate Terminal Drivers

In This Chapter

This chapter discusses how alternate terminal drivers fit into the overall CA-Clipper architecture as well as how to install and use each of the supplied terminal drivers. The following major topics are discussed:

- The Alternate Terminal Driver Architecture
- The ANSITERM Alternate Terminal Driver
- The NOVTERM Alternate Terminal Driver
- The PCBIOS Alternate Terminal Driver

The Alternate Terminal Driver Architecture

CA-Clipper supports a driver architecture that allows CA-Clipper-compiled applications to use alternate terminal drivers. This architecture provides support for nonstandard video hardware and ANSI output devices, allowing your applications to run in a wider variety of environments.

The following terminal drivers are supplied as part of the CA-Clipper Development System and are discussed in this chapter:

- The ANSITERM driver provides ANSI terminal support for systems that require it
- The NOVTERM driver causes CA-Clipper applications to execute faster when run on some nondedicated network server software
- The PCBIOS driver provides direct BIOS calls rather than direct screen writes for systems requiring this form of I/O

In CA-Clipper, communication with I/O devices is controlled by a multilayered terminal system. At the lowest level is the *terminal driver* which controls screen and keyboard activity. It consists of a screen and keyboard driver that communicates directly with the I/O device (operating system or hardware). It is the device-specific part of the CA-Clipper terminal system.

There is, then, a higher level system that communicates with terminal drivers. This system is known as the *General Terminal* (GT) system and provides general services that create CA-Clipper screen and keyboard commands and functions. The following figure demonstrates:

CA-Clipper screen and keyboard commands and functions	
General Terminal (GT.OBJ)	
Terminal Driver	
Screen	Keyboard

The default terminal driver, designed for IBM PC and 100 percent compatibles, is supplied as a library file (TERMINAL.LIB) installed into your \CLIP53\LIB directory. This driver links into each program automatically if you specify no alternative terminal driver, provided that you do not use the /R option when you compile. An alternate terminal driver is supplied as a separate library (.LIB) file that links into an application program in place of the default terminal driver, if you specify it on the link line.

All alternate terminal drivers work through the General Terminal layer as supplied in the file GT.OBJ. The CA-Clipper installation program installs this file in the \CLIP53\OBJ subdirectory on the drive that you specify, so you need not install the driver manually.

The ANSITERM Alternate Terminal Driver

The ANSITERM terminal driver supports the ANSI screen mode for all screen display from CA-Clipper programs.

This screen mode is installed by specifying ANSI.SYS in the user's CONFIG.SYS. ANSI.SYS replaces the default DOS CON device driver for video display and keyboard input. Once installed, it supports ANSI escape sequences to erase the screen, set the screen mode, and control the cursor in a hardware-independent way. Most modern DOS programs, however, do not use it and write either directly to the video hardware or use BIOS routines for enhanced screen performance.

Use the ANSI screen mode for CA-Clipper programs that run on hardware that does not support either writing to video hardware or BIOS calls for screen display. This is the case when using alternative display hardware to support the blind.

Note: The ANSITERM terminal driver fully supports all screen and keyboard functionality of the default terminal driver. This includes the ability to save and restore screens, as well as support for all keys on the standard 101-key keyboard.

Installing ANSITERM Terminal Files

The ANSITERM terminal driver is supplied as the file ANSITERM.LIB. The CA-Clipper installation program installs this file in the \CLIP53\LIB subdirectory on the drive that you specify, so you need not install it manually.

Linking the ANSITERM Terminal Driver

To link the ANSITERM alternate terminal driver into an application program, you must specify both GT.OBJ and ANSITERM.LIB to the linker along with your application object (.OBJ) modules.

To link with the protected mode linker, use:

```
EXOSPACE FI <appObjectList>, GT LI ANSITERM
```

Note: These link commands assume you have set the LIB and OBJ environment variables to the standard locations. They also assume that the CA-Clipper programs were compiled without the /R option.

The Runtime Environment

Using ANSITERM.LIB requires that ANSI.SYS be installed on the user's computer. To accomplish this, include the following statement in the user's CONFIG.SYS:

```
DEVICE=ANSI.SYS
```

Performance Concerns

Because the ANSITERM terminal driver uses buffered screen writes for all screen painting, some operations, especially those that scroll the screen, are slow. These include:

1. All box drawing commands and functions
2. All console commands and functions when scrolling
3. All clear screen commands and functions
4. All restore screen commands and functions
5. Standard out functions (OUTSTD() and OUTERR()) whether the screen is scrolling

Note: Overall performance of CA-Clipper programs is slower since the ANSITERM terminal driver must spend more time polling for user events than the standard CA-Clipper terminal driver.

Screen Output from C and Assembly Language

The ANSITERM terminal driver overwrites all output from C and Assembly Language when it refreshes the screen from the screen buffer. As a consequence, you should perform all screen output from CA-Clipper.

The ANSITERM terminal driver also virtualizes the cursor. This means that BIOS functions that report the location of the hardware cursor will not always return the correct value. To obtain the cursor position, use the CA-Clipper ROW() and COL() functions instead.

Other Incompatibilities

1. ISCOLOR() always returns false (.F.).
2. When you load DBU, the default color mode is monochrome unless you specify DBU with the /C command line option.
3. The first time you invoke the debugger, the default color mode is also monochrome unless you set the Options Mono display off.
4. When an application linked with the ANSITERM terminal driver terminates, the last color set in the application becomes the DOS color. This happens because colors set with ANSITERM are global to DOS and CA-Clipper cannot query DOS for the current screen colors as the application loads.
5. Nondisplaying ASCII characters are presented as a space by the ANSITERM terminal driver. These include BELL (CHR(7)), BS (CHR(8)), TAB (CHR(9)), LF (CHR(10)), CR (CHR(13)), and ESC (CHR(27)).

The NOVTERM Alternate Terminal Driver

The NOVTERM terminal driver is a special-purpose driver that circumvents an incompatibility between some nondedicated network server software and CA-Clipper. This incompatibility causes printers connected to the server to slow to an unusable rate.

CA-Clipper applications and nondedicated servers compete for resources. CA-Clipper applications make use of the time between keystrokes to perform various system tasks. This greatly improves the application's overall performance by limiting its idle time. Certain nondedicated servers only attempt to print within an application's idle time. Since a CA-Clipper application is seldom idle, this greatly slows printing.

***Important!** The NOVTERM terminal driver corrects the incompatibility by preventing the CA-Clipper application from using idle time. Because this can severely hamper performance, you should only use the NOVTERM terminal driver when necessary, and then you should link it only into those applications that are physically running the nondedicated server.*

Note: The NOVTERM terminal driver fully supports all screen and keyboard functionality of the default terminal driver. This includes the ability to save and restore screens, as well as support for all keys on the standard 101-key keyboard.

Installing NOVTERM Terminal Files

The NOVTERM terminal driver is supplied as the file NOVTERM.LIB. The CA-Clipper installation program installs the driver file in the \CLIP53\LIB subdirectory on the drive that you specify, so you need not install it manually.

Linking the NOVTERM Terminal Driver

To link the NOVTERM alternate terminal driver into an application, you must specify both GT.OBJ and NOVTERM.LIB to the linker with your application object (.OBJ) modules.

To link with the protected mode linker, use:

```
EXOSPACE FI <appObjectList>, GT LI NOVTERM
```

Note: These link commands assume you have set the LIB and OBJ environment variables to the standard locations. They also assume that the CA-Clipper programs were compiled without the /R option.

Performance Concerns

Overall performance of CA-Clipper programs is slower, since the NOVTERM terminal driver must spend more time polling for user events than the standard CA-Clipper terminal driver and since the program will not use its idle time for other tasks.

Screen Output from C and Assembly Language

The NOVTERM terminal driver overwrites all output from C and Assembly Language when it refreshes the screen from the screen buffer. Therefore, you should perform all screen output from CA-Clipper.

The NOVTERM terminal driver also virtualizes the cursor. This means that BIOS functions that report the location of the hardware cursor will not always return the correct value. To obtain the cursor position, use the CA-Clipper functions—ROW() and COL().

The PCBIOS Alternate Terminal Driver

The PCBIOS terminal driver uses BIOS calls instead of direct screen writes. It is designed for applications that trap BIOS calls to redirect output over telecommunication lines or to convert output to a form compatible with two-byte character sets.

Note: The PCBIOS terminal driver fully supports all screen and keyboard functionality of the default terminal driver. This includes the ability to save and restore screens, as well as support for all keys on the standard 101-key keyboard.

Installing PCBIOS Terminal Files

The PCBIOS terminal driver is supplied as the file, PCBIOS.LIB. The CA-Clipper installation program installs the driver file in the \CLIP53\LIB subdirectory on the drive that you specify, so you need not install it manually.

Linking the PCBIOS Terminal Driver

To link the PCBIOS alternate terminal driver into an application program, you must specify both GT.OBJ and PCBIOS.LIB to the linker in addition to your application object (.OBJ) modules.

To link with the protected mode linker, use:

```
EXOSPACE FI <appObjectList>, GT LI PCBIOS
```

Note: These link commands assume you have set the LIB and OBJ environment variables to the standard locations. They also assume that the CA-Clipper programs were compiled without the /R option.

Performance Concerns

Because the PCBIOS terminal driver uses buffered screen writes for all screen painting, some operations, especially those that scroll the screen, are slow. These include:

1. All box drawing commands and functions
2. All console commands and functions when scrolling
3. All clear screen commands and functions
4. All restore screen commands and functions
5. Standard out functions (OUTSTD() and OUTERR()) whether the screen is scrolling

Screen Output from C and Assembly Language

The PCBIOS terminal driver also overwrites all output from C and Assembly Language when it refreshes the screen from the screen buffer. Therefore, you should perform all screen output from CA-Clipper.

The PCBIOS terminal driver also virtualizes the cursor. This means that BIOS functions that report the location of the hardware cursor do not always return the correct value. To obtain the cursor position, use the CA-Clipper functions—ROW() and COL().

Summary

This chapter has introduced you to the alternate terminal driver concept, giving you specific information on the architecture used to implement them in CA-Clipper. Each of the alternate terminal drivers supplied with CA-Clipper was discussed, including how to link and use it into your application and the implications of doing so.

Index