

NIAKWA DEVELOPMENT TOOLS

DATA MANAGER

PROGRAMMER'S GUIDE

1st. Edition - June 1991
COPYRIGHT © 1991 Niakwa, Inc.

Niakwa, Inc.
23600 N. Milwaukee Avenue
Mundelein, IL. 60060

Tel: (708) 634-8700 FAX: (708) 634-8718 TELEX: 3719965 NIAK UB

DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITIES AND PROPRIETARY RIGHTS

The staff of Niakwa, Inc. (Niakwa), has taken due care in preparing this manual. Nothing contained herein shall be construed to modify or alter in any way the standard terms and conditions of the Basic-2C Support and Distribution License Agreement, End-User Support Only License Agreement, the Niakwa Software License Agreement and Warranty, or any other Niakwa License Agreement (collectively the "License Agreements") by which this software package was acquired.

This manual is to serve as a guide for use of the Niakwa software only and not as a source of representations or additional undertakings by Niakwa. The licensee must refer to the License Agreements for Niakwa product and service representations.

No ownership of Niakwa Software is transferred by any of the License Agreements. Any use of Niakwa software beyond the terms and conditions of the License Agreements, without the written authorization of Niakwa is prohibited.

Niakwa is a registered trademark of Niakwa Management Services 1975 Ltd., and is licensed to Bluebird Systems.

Basic-2C, BLUEBIRD, SuperDOS, and SDtrieve are registered trademarks of Bluebird Systems.


All other trademarks are property of their respective holders.

PREFACE

The Niakwa Data Manager (NDM) allows Basic-2C applications to use state of the art native ISAM products to store data while retaining full portability. This section describes the organization and use of the NDM manuals, the required knowledge necessary for use of the NDM, the notational conventions used to denote command codes and parameters, and the associated documents that may be needed with this manual.

Notational Conventions

The NDM Programmer's Guide and Platform Specific Addendums use the following notational conventions.

N
C  **TE:** **Notes provide information of particular importance.**

Warning are special conditions that require special care by the user. Disregarding this information could result in a serious problem.

Examples in this manual use both the number and name of the API functions. The API function name is used for documentation purposes only. API function names should not be used in actual code, only the API function numbers.

Long variable names used for the API parameters are for documentation purposes only. When actually programming, use the standard Basic-2C variable or literal formats. All alpha parameters used in API function descriptions will end in \$ (MODE\$\$) while numeric parameters do not (RETURN_CODE). For example:

31060 NDM_CREATE_FILE (FILE_NAMES\$, MODE\$, INDEX_NUMBER, KEY-DESC_TABLE\$, RECORD_LENGTH, ISAM_SPECIFICS, FILE_HANDLES\$, RETURN_CODE)

Would be coded as:

```
GOSUB' 31060 ("EXAMPLE", M$, I, K$, L, A$, H$, R)
```

Table of Contents

PREFACE

INTRODUCTION

Overview.....	1 - 1
Product Concepts.....	1 - 2
Product Organization	1 - 3
Data Manager Features.....	1 - 4
Product Benefits.....	1 - 5
Prerequisite Knowledge	1 - 6
NDM Documentation Overview	1 - 6

NDM CONCEPTS

Overview.....	2 - 1
Basic-2C NDM API Access.....	2 - 2
ISAM Concepts.....	2 - 3
File and Record Orientation	2 - 3
Records Accessed By Indices	2 - 3
Current Record.....	2 - 4
Index Concepts.....	2 - 5
Current Index.....	2 - 5
Index Segments	2 - 6
Index Field Types	2 - 6
Ascending Versus Descending Segments.....	2 - 7
Unique Versus Non-Unique Indices.....	2 - 7
Defining Indices	2 - 8
Permanent Indices.....	2 - 8
Dynamically Creating and Deleting Indices.....	2 - 9
Maintenance of Indices.....	2 - 9
Index Limits.....	2 - 9
Return Codes.....	2 - 10

DATA DICTIONARY FILES

Overview.....	3 - 1
---------------	-------

Catalog File 3 - 2
 Advantages of Use 3 - 2
 Layout of Catalog File 3 - 3
 Data Description 3 - 5
 Data Description File Layout 3 - 6
 Date Field Types 3 - 11
 Key Description File 3 - 12
 Key Description File Layout 3 - 13
 Relation File 3 - 15
 Relation File Format 3 - 15

NDM UTILITIES

Overview 4 - 1
 Starting the NDM Utilities 4 - 2
 The Utility Main Menu 4 - 3
 NDM Utility Operation 4 - 6
 Main View Screen Operation 4 - 6
 Catalog File Maintenance 4 - 7
 Features 4 - 8
 General Use 4 - 8
 Error Description File Maintenance 4 - 12
 Features 4 - 12
 General Use 4 - 12
 Field Type File Maintenance 4 - 15
 Features 4 - 15
 General Use 4 - 17
 Data Description File Maintenance 4 - 17
 Features 4 - 18
 General Use 4 - 18
 Creating A New Data Description 4 - 19
 Modifying/Viewing a Data Description File 4 - 22
 Data Description Reorganization 4 - 23
 Key Description File Maintenance 4 - 24
 Features 4 - 24
 General Use 4 - 24
 Creating a New Key Description File 4 - 25
 Modifying/Viewing a Key Description File 4 - 28
 Access User Data Files 4 - 30
 Features 4 - 30
 General Use 4 - 30
 Copy User Data File 4 - 35

Table of Contents

Features	4 - 35
General Use	4 - 35
Export Data Dictionary to IQ Utility	4 - 38
Features	4 - 38
General Use	4 - 38
Convert Native File to Basic-2C Format.....	4 - 42
Features	4 - 42
General Use	4 - 42
Convert Basic-2C Format Files to Native File.....	4 - 43
Features	4 - 43
General Use	4 - 43
Stage 1 of Conversion Process	4 - 44
Stage 2 of Conversion Process	4 - 45
Set NDMUTIL Options.....	4 - 45
Features	4 - 45
General Use	4 - 46
Install New Catalog File	4 - 47
Features	4 - 47
General Use	4 - 47
Display Program Information.....	4 - 49
Features	4 - 49
General Use	4 - 49
Translation File Maintenance.....	4 - 50
Features	4 - 50
General Use	4 - 51

NDM ADVANCED CONCEPTS

Overview.....	5 - 1
Multi-User Considerations	5 - 2
File Locking.....	5 - 2
Record Locking	5 - 2
Managing File Handles	5 - 4
Maintaining Application Portability.....	5 - 4
Concept of Portability.....	5 - 4
File Naming and Location	5 - 5
Native Field Types.....	5 - 5
Common Native ISAM Features.....	5 - 6
Field Type Conversions.....	5 - 7
The Conversion Table	5 - 8
Establishing the Conversion Table	5 - 8
When To Call 31030 NDM_CONVERT.....	5 - 11

Performance Issues	5 - 12
Managing Conversion Table Handles.....	5 - 13
Toolbox Features	5 - 14
Filename Extensions	5 - 14
Key Types	5 - 14
File Limits.....	5 - 15
Transaction Processing.....	5 - 15
Get/Set Position	5 - 16
Create/Delete Index.....	5 - 16
Transaction Tracking	5 - 16
Transaction Tracking Statements.....	5 - 17

NDM PROGRAMMING

Overview.....	6 - 1
Basic NDM Statements	6 - 4
Startup Routines	6 - 6
Determining Parameter Sizes	6 - 7
Initialization of NDM.....	6 - 8
Get Default NDM Directory.....	6 - 9
Open Catalog File.....	6 - 9
Enable Extended Features	6 - 9
Define Variables	6 - 9
Sample Program for Startup Routines.....	6 - 10
Shutdown Routines.....	6 - 11
Creating NDM Data Files	6 - 11
Create File Example 1.....	6 - 12
Create File Example 2.....	6 - 12
Creating Files with No Data Description or Key Description Files.....	6 - 13
Opening a File	6 - 14
Selecting an Index	6 - 16
Reading a Record by Key	6 - 17
Reading Records Sequentially	6 - 18
Reading from Logical Start of File.....	6 - 19
Reading Sequentially from a Starting Key Value	6 - 20
Adding New Records	6 - 20
Updating Records	6 - 21
Deleting Records	6 - 22

SYSTEM AND SUPPORT FILES

Overview.....	7 - 1
---------------	-------

Table of Contents

Data Dictionary System Files	7 - 2
NDM Support Files	7 - 3
Field Type File	7 - 3
Error File	7 - 5
Translation Table File	7 - 6

CONVERTING APPLICATIONS AND DATA FILES

Overview.....	8 - 1
Guidelines for Program Code Conversions.....	8 - 2
Use of NDM Data Dictionary Files	8 - 2
Use of Field Type Conversion	8 - 3
Use of Modular Code	8 - 3
Use of Indices	8 - 4
\$OPEN/\$CLOSE.....	8 - 4
Record Locking.....	8 - 5
Converting Specific Application Types.....	8 - 6
Applications that use Generalized Keyed Record Access.....	8 - 6
Applications that use Generalized Routines to Return Pointers	8 - 7
DC Style Applications.....	8 - 7
Converting Existing Data Files.....	8 - 8
Converting Data Dictionary Files	8 - 9

NDM EXAMPLE PROGRAMS

Overview.....	9 - 1
Starting the Example Programs.....	9 - 2
Simple Examples Using API Functions	9 - 3
NDM Example One (EX1NDM.OBJ).....	9 - 3
NDM Example Two (EX2NDM).....	9 - 5
NDM Example Three (TESTCUST).....	9 - 7
Advanced Program Support Files	9 - 8
Advanced Program General Operation	9 - 8
Create Customer File	9 - 9
Open Customer File	9 - 9
Add New Records	9 - 10
Modify Records	9 - 11
Delete Records	9 - 11
Read Records by Key.....	9 - 11
Read Records From Logical Start of File	9 - 12
Read Records From a Starting Key Value	9 - 13
QUIT.....	9 - 13

Advanced Program Main Features 9 - 14
 Advanced Program Operation..... 9 - 14

API FUNCTIONS

Overview..... 10 - 1
 API Function Calls..... 10 - 3
 API Function Listing in Numeric Order 10 - 4

RETURN CODES

Overview..... 11 - 1
 Return Code Listing 11 - 2
 Return Code File..... 11 - 5
 Explanation of Return Codes 11 - 5

DATA DICTIONARY FILE CREATION TUTORIAL

Overview..... L - 1
 Data Dictionary File Creation Guidelines L - 2
 Creating the Catalog File L - 3
 Creating the Data Description File..... L - 4
 Creating the Key Description File L - 8
 Creating a Catalog Entry for the Data File..... L - 12
 Adding/Modifying Data File Records..... L - 15
 Converting an NDM Data Dictionary to IQ..... L - 19

NDM INTERNAL TABLES

Overview..... B - 1
 NDM Key Description Table Format B - 2
 Limits Table B - 3
 Native ISAM Specific Table Format B - 3

GLOSSARY

CHAPTER 1

INTRODUCTION

1.1 Overview

The Niakwa Data Manager (NDM) provides Basic-2C programmers with sophisticated, fully portable, ISAM capabilities. The NDM products allow Basic-2C applications to use state of the art native ISAM products to store data while retaining full portability. Although modifications to existing applications are required to use these features, the NDM product is designed to minimize these modifications.

Section 1.2 discusses the NDM product concept.

Section 1.3 discusses the NDM product organization.

Section 1.4 discusses the NDM features.

Section 1.5 discusses the benefits of the NDM product.

Section 1.6 discusses the prerequisite knowledge necessary to use the NDM.

Section 1.7 provides an overview of the NDM documentation.

NOTE: Appendix A provides a tutorial for creating the catalog, data description, and key description files. It is recommended that these files be created before any NDM application is used.

1.2 Product Concepts

The NDM products are based on the external call feature of Basic-2C. For each hardware/operating system platform supported by Basic-2C, an interface to one or more native ISAM products (selected by Niakwa) is provided. An Application Program Interface (API) designed by Niakwa is also provided. This API consists of a set of sub-routines that can be called by the application program by using simple GOSUB statements.

For example:

```
GOSUB' 31370 (F$,R$,P,L$,R)
```

may be used to read a record.

The interface to the native ISAM, the API, is provided as a pre-compiled, pre-linked external library to avoid variable and line-number conflicts with existing applications. Thus, application developers have no need to compile or link any routines and knowledge of any language other than Basic-2C is not required (refer to Figure 1-1).

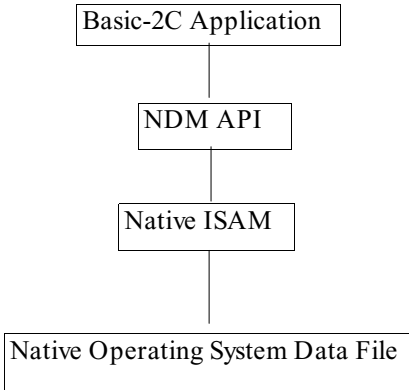


Figure 1 - 1

1.3 Product Organization

An NDM product is available for each platform supported by Basic-2C Release III or greater. Each platform specific product consists of an NDM Development Package and an NDM RunTime Package that is required for every end user site.

For each platform, the NDM Development Package contains:

- Platform specific and native ISAM specific installation and operation documentation.
- A set of utilities to allow for creation and maintenance of the NDM data dictionary files: catalog, data description, key description, and relation.
- A set of routines that allow developers who use their own external routines to integrate those routines with the NDM Development Package.

For each platform, the NDM RunTime Package contains:

- An external library containing the Niakwa API and an interface to a supported native ISAM.
- End user level installation instructions.

1.4 Data Manager Features

The NDM provides the following data management features:

- File locking and record locking are both supported.
- All data is stored in stand alone native operating system files. These files are expandable.
- Read by key functions to search for specified key values that support seek capabilities: equal, greater than, less than, greater than or equal to, and less than or equal to.
- Position functions allowing read next, read previous, read first and read last.
- Transaction start, end, and abort functions. These functions allow operations to be logically grouped into transactions such that partially completed transactions can be reversed upon an error or hardware failure.
- Data conversion functions are provided to allow easy conversion between Basic-2C field types and field types supported by the native ISAM. Utilization of these functions is not required but is recommended to ensure that the data is accessible by various third party products.
- A "toolbox" feature that allows support for specific native ISAM functions not supported by all native ISAMs.

- Multiple indices:

Up to 9 keys can be defined.

Each key may contain up to 8 segments.

Up to 24 key segments per file may be defined.

1.5 Product Benefits

The NDM provides significant improvements in functionality over the data management function built into Basic-2C.

Performance improvements are generated in several ways. These include:

1. Elimination of the \$OPEN bottleneck. Since each data file is now a stand alone native operating system file, the fact that one user has one file locked does not prevent a second user from accessing other files.
2. Support for record locking. Applications may reduce or eliminate the need for file locking, thus allowing multiple users to concurrently update data in different records within the same file.
3. Buffering and caching performed by the native ISAM. This typically improves performance significantly.
4. Applications may improve performance significantly by using the improved native ISAM capabilities.

Data Independence

Data stored using the Niakwa API is accessible to any other Basic-2C application that uses the API, thus providing data independence. In addition, the data is directly accessible by **any** third party product that supports the native ISAM in use (i.e., Intelligent Query, Focus, Informix, etc.).

Programmer Productivity

Application development and maintenance is greatly simplified since the native ISAM, not the application, manages the data.

1.6 Prerequisite Knowledge

A good working knowledge of Basic-2C is essential before attempting to use the NDM.

The native ISAM must be loaded and configured on the host platform before attempting to use the NDM.

NOTE: A thorough knowledge of the programming interface of the native ISAM is not required. However, installation and operational considerations for the native ISAM must be fully understood by the developer. Refer to Chapter 2 of the Platform Specific Addendum for more information on this subject.

The developer should also be familiar with the native operating system operation.

No knowledge of languages other than Basic-2C is required. All NDM terms are defined in the Glossary found at the end of this Programmer's Guide.

1.7 NDM Documentation Overview

The NDM documentation is organized into two manuals. The first is the NDM Programmer's Guide. This describes the NDM features generic to all platforms and native ISAMs. The second manual is the Platform Specific Addendum. This covers the installation and configuration of the NDM and specific NDM features for a particular platform and native ISAM.

The Programmer's Guide is organized by chapters where the first chapter serves as an introduction to the NDM and the NDM documentation.

Chapter 2 provides detailed information on the concepts behind the NDM: API access from Basic-2C, ISAMs, indices, and return codes.

Chapter 3 introduces the NDM data dictionary files.

Chapter 4 provides documentation on the NDM Utilities that are provided with the NDM.

Chapter 5 provides information on the advanced concepts of the NDM including multi-user capabilities, portability, field type conversions, performance issues, the toolbox feature, and transaction tracking.

Chapter 6 provides details on NDM programming.

Chapter 7 discusses the various NDM system and support files.

Chapter 8 discusses the conversion of existing Basic-2C applications and data files for use with the NDM.

Chapter 9 describes the example programs provided with the NDM Development Package.

Chapter 10 details the NDM function calls.

Chapter 11 provides details on all NDM return codes.

Appendix A provides a tutorial of the creation of the data dictionary files for the example program discussed in Chapter 9.

Appendix B provides layouts for internal NDM tables.

The Glossary provides the definitions of the terminology used in the NDM documentation and programs.

This manual explains the use of the NDM product. Refer to the Platform Specific Addendum for information regarding the installation and use of NDM with specific operating systems and native ISAM's. Refer to the Basic-2C Technical Reference Guide and Hardware Supplement for Basic-2C considerations and to the native ISAM manuals for native ISAM considerations.

CHAPTER 2

NDM CONCEPTS

2.1 Overview

This chapter details the concepts used by the NDM product.

Section 2.2 discusses how the NDM API is accessed through Basic-2C statements.

Section 2.3 discusses ISAM concepts.

Section 2.4 discusses index concepts and keys.

Section 2.5 discusses defining indices.

Section 2.6 discusses return codes.

NOTE: NDM terminology is defined in the Glossary found at the end of this Programmer's Guide.

2.2 Basic-2C NDM API Access

This section briefly reviews the Basic-2C statements that are used to call the NDM

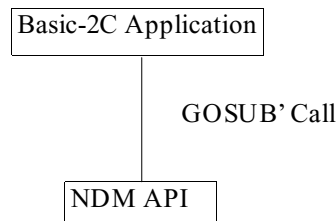


Figure 2 - 1

API functions. To access any API function from within a Basic-2C program, it is necessary to use the **GOSUB'** statement (refer to the Basic-2C Statements Manual for the syntax of this statement). The API functions are accessed through external calls. All NDM API functions are numbered from **31010** to **31460** (see Figure 2-1).

The following is an example NDM call:

```
0010 GOSUB' 31320 (10,200,10,1024,R)
```

Upon execution of the **GOSUB'**, control is transferred to the external API routine. Upon completion of the external API routine, control is transferred to the statement immediately following the **GOSUB'** statement. This is functionally identical to the internal **DEFFN'** (refer to the Basic-2C Statements Manual for more information on the **DEFFN'** statement). There is one important difference between using internal **DEFFN'**s and external **DEFFN'**s. With internal **DEFFN'**s, any values to be returned by the **DEFFN'** must be set in variables and examined by the calling routine. With external **DEFFN'**s, values to be returned are placed in variables that are part of the parameter list passed to the **DEFFN'**.

In the example above, the variable **R** is passed as a parameter to **'31320** (this parameter is the return code that is set by the **'31320** routine). Thus, the value of **R** may be examined after the **GOSUB'** statement.

NOTE: For detailed documentation of each NDM API subroutine, refer to Chapter 10 of this Programmer's Guide. Chapter 10 also clearly defines each API function's input and output parameters.

2.3 ISAM Concepts

The NDM provides an API to multiple native ISAMs. The native ISAM does all the actual data management for the NDM. The following section details the various ISAM concepts used by the NDM.

2.3.1 File and Record Orientation

All native ISAM products supported by the NDM, and therefore the NDM itself, are record oriented access methods. This means that the logical unit of data managed by the NDM is a physical record within a physical file where the file typically contains a single record type.

2.3.2 Records Accessed By Indices

The NDM deals with the logical order of records. Logical order is determined by indices defined by the developer.

NOTE: Physical order of the records is not relevant to the NDM and is handled by the native ISAM.

Records along a specified index can be retrieved by two basic methods:

- 1) By key value using **31360 NDM_READ_BY_KEY**. This function allows specification of an exact key value to be searched for or specification of an equivalence operator (equal to, greater than, greater than or equal to, less than, less than or equal) for searches based on inexact keys. The NDM returns the first record, if any, matching the specified key value according to the specified equivalence.

- 2) By logical position using **31370 NDM_READ_BY_POSITION**. This function allows sequential movement along the logical path specified by the current index. Movement may be forward or backward (one record at a time) through the file, or movement to the logical BOF (record with first key value for the current index) or logical EOF (record with last key value for the current index) may be specified.

2.3.3 Current Record

The NDM internally maintains a pointer to the current record for each file. The current record is used for several NDM operations. These include:

31120 NDM_DELETE_RECORD deletes the current record. After a record has been deleted, current record still points to that record. This allows **31370 NDM_READ_BY_POSITION** with values 1 (next record) or -1 (previous record) to be used, but attempts to read the current record will fail.

31370 NDM_READ_BY_POSITION changes the current record based on the position specification and then returns the new current record. Changes in position to the next or previous record are based on the current record value prior to the call.

31380 NDM_REWRITE_RECORD rewrites the current record and leaves the current record set to the record rewritten.

The current record is set to undefined whenever a new index is selected either explicitly by **31400 NDM_SET_CURRENT_INDEX** or implicitly by **31340 NDM_OPEN_FILE**, **31060 NDM_CREATE_FILE**, **31070 NDM_CREATE_FILE_FROM_CATALOG** or **31080 NDM_CREATE_INDEX**.

When the current record is undefined, any operations that require the current record to be set will return an error code. The following operations can be used to set the current record when it is undefined:

31360 NDM_READ_BY_KEY changes the current record to the first matching record found, if any. If no match is found, the current record is set to EOF for forward searches, BOF for backward searches, or undefined for exact matches.

31370 NDM_READ_BY_POSITION with a position specification of BOF or EOF. This sets the current record to BOF or EOF respectively. Any other specifi-

cation, such as forward one position or backward one position results in an error code when the current record is not previously defined.

2.4 Index Concepts

The NDM keeps track of a current index for each open data file. Indices are defined when a file is created or may be defined later by **31080 NDM_CREATE_INDEX**.

2.4.1 Current Index

The current index is a numeric value associated with each open file that identifies the index currently in use for that file. Its value affects the API functions that alter the current record, such as **31370 READ_BY_POSITION**.

The Current Index value is set by:

- **31060 NDM_CREATE_FILE**
- **31070 NDM_CREATE_FILE_FROM_CATALOG**
- **31080 NDM_CREATE_INDEX**
- **31340 NDM_OPEN_FILE**
- **31400 NDM_SET_CURRENT_INDEX**.

NOTE: An application can determine the value of the current index by calling the **31180 NDM_GET_CURRENT_INDEX** function.

If set to zero, the current index and the current record number for that file are undefined. Consequently, any operations that use these values return an error code. Here, only certain operations, such as **31220 NDM_GET_FILE_STATUS**, will be available for the file. The current index may be reset to a usable value by calling **31400 NDM_SET_CURRENT_INDEX** with a valid index number.

2.4.2 Index Segments

An index must contain one or more segments. Segments need not be contiguous within the record. Segments are usually associated with a specific defined field, but this is not necessarily the case. Partial fields can be defined as index segments by the application either by building the key description table internally or by defining partial fields in the data description file.

For example, Table 2-1 shows a data description file containing partial field definitions where I-Name is a partial field corresponding to the first 5 characters of Name. This shorter segment may improve performance in data file access.

Field	Conversion Table	Start	Length
Name	0	1	20
I-Name	1	1	5

Table 2-1

Table 2-2 provides another example of use of partial fields as index segments. Assume that the Phone-Number field is in the format (xxx)-yyy-yyyy, where xxx is the area code. Defining Area-Code as a segment allows indexing by area code while preserving Phone-Number as a single field.

Field	Conversion Table	Start	Length
Phone-Number	0	1	13
Area-Code	1	2	3

Table 2-2

2.4.3 Index Field Types

For an index to work properly, the native ISAM must be able to understand the contents of the fields that comprise the index. It must be able to collate the data in these fields properly. Different native ISAMs support different field types. The only field types that are supported by all native ISAMs as indices are string (equivalent to Basic-2C alphanumeric), integer, date, and time.

NOTE: It is recommended that applications limit their use of indices to these field types. However, for applications that must use other field types for special purposes, the

31420 NDM_SET_TOOLBOX_STATUS function can be used to enable this capability.

Refer to the Platform Specific Addendum for details on field types supported as indices by the native ISAM.

2.4.4 Ascending Versus Descending Segments

Ascending segments are segments that increase in value (i.e., 1, 2, 3; a, b, c; etc.) while descending segments decrease in value (i.e., 3, 2, 1; c, b, a; etc.).

It is strongly recommended that segments be ascending. However, there are special cases where descending segments are necessary. For example, when printing a report based on the customer's area codes and year-to-date sales, if the area codes are to be printed in ascending order and YTD-Sales printed in descending order within each area code, the index would be defined as follows:

```
Index Name: Sales - (non-unique index)
Segment 1 - Area-Code - Ascending
Segment 2 - YTD-Sales - Descending
```

If this index is selected, reading records sequentially will yield all customers in the lowest Area-Code ordered by YTD-Sales with the customers with the greatest YTD-Sales listed first. This is followed by all the customers in the second lowest Area-Code ordered by YTD-Sales in descending order, and so on.

If ordering customers by Area-Code was not required and YTD-Sales was set up as an ascending key, customers with the highest YTD-Sales could still be retrieved first. To do this, position the current record pointer to EOF, and then read sequentially backward through the file.

2.4.5 Unique Versus Non-Unique Indices

Each index must be defined as being either unique or non-unique. This definition is made at the time the index is created (either by **31060 NDM_CREATE_FILE**, **31070 NDM_CREATE_FROM_CATALOG** or **31080 NDM_CREATE_INDEX**). Defining an index as unique means that duplicate key values are not allowed for that index and that any attempt to add a record or rewrite a record that contains a value in the index that already exists in any other record in the file will result in an error (NDM error code 7).

NOTE: Under NDM, any index may be defined as non-unique. There is no requirement for a "primary"unique index.

For example, as a key field, zip code, could be a non-unique key because many customers can have the same zip code. A customer's account number would typically be a unique key, since two customers should not have the same account number.

The order of duplicate values for non-unique indices is not defined. If a particular order is required, define a multi-segment index. Refer to Section 2.4.2 for more information.

2.5 Defining Indices

The NDM allows for definition of both permanent and temporary indices.

2.5.1 Permanent Indices

When a file is created, a description of the indices must be passed to the native ISAM. This description is called the key description table. The key description table can be built in one of two ways:

- 1) Use of the key description file with the data description file created by the NDM Utilities allows developers a method of defining indices with no programming.

NOTE: The 31090 NDM_CREATE_KEY_TABLE function constructs a key description table based on the information contained in the key description and data description files. When using 31060 NDM_CREATE_FILE to create the file, 31090 NDM_CREATE_KEY_TABLE must be called explicitly by the application. 31070 NDM_CREATE_FILE_FROM_CATALOG calls 31090 NDM_CREATE_KEY_TABLE implicitly.

- 2) If the developer chooses not to use the key description file, then the application must construct the key description table itself and pass this to **31060 NDM_CREATE_FILE** when creating a file. The structure of the key description table is defined in Appendix B.

2.5.2 Dynamically Creating and Deleting Indices



The **31080 NDM_CREATE_INDEX** function allows for the dynamic creation of indices. This adds a new index to the list of indices attached to the specified open data file. The developer must build the key description table to use this API function. Refer to Appendix B for details.

The most common reason for using **31080 NDM_CREATE_INDEX** is for sorting data in an application. When the application has finished using the index, it can be deleted using **31110 NDM_DELETE_INDEX**.

NOTE: **31110 NDM_DELETE_INDEX** may be used to delete any index. However, Niakwa strongly recommends that this be used only to delete temporary indices.

*Changes made to a file's index structure via **31080 NDM_CREATE_INDEX** and **31110 NDM_DELETE_INDEX** are NOT automatically reflected in the key description file.*

2.5.3 Maintenance of Indices

The native ISAM maintains all indices automatically. Utilities to rebuild damaged indices are typically included with the native ISAM.

2.5.4 Index Limits

Table 2-3 discusses the limits that are imposed on NDM indices. Depending on the native ISAM, these limits can be extended by feature number 3 of **31420 NDM_SET_TOOL_BOX_STATUS**.

NDM indices	Limits
Total number of indices	9
Segments per key	8
Total number of segments	24
Total length of key	120 bytes

Table 2-3

NOTE: Going beyond the NDM limits and using the toolbox features may affect an application's portability.

2.6 Return Codes

Every API function sets a return code value. If this value is anything other than zero, an error has occurred. Thus, the return code should be checked by the application after every API function call for non-zero values.

The return codes issued will be the same from one platform and native ISAM to the next. Therefore, the application can, and should, check for specific error code values and take action accordingly. Some of these error codes are listed in Table 2-4.

NDM Return Code	Description	Comment
1	End of File	This code is typically generated when the application is reading forward through a file.
2	Beginning of File	This code is typically encountered when an application is reading backward through a file.
6	Key Value Not Found	This can be encountered during a 31360 NDM_READ_BY_KEY call.
7	Duplicate Keys	This can be encountered during a 31330 NDM_INSERT_RECORD and 31380 NDM_REWRITE_RECORD call if there is an index defined that does not allow duplicate entries.
42	Record in use	This can be encountered during any call that attempts to access a record that is locked by any terminal - except the terminal issuing the request. Typically, the application should retry the operation until it succeeds.
43	File in use	This can be encountered during 31340 NDM_OPEN_FILE if an attempt is made to open the file in "Exclusive Mode" while other terminals have the file open or whenever any other terminal has the file open in exclusive mode.

NDM Return Code	Description	Comment
48	Permission Error	This can be encountered during 31340 NDM_OPEN_FILE if the user does not have proper privileges to open the specified file.

Table 2-4

NOTE: Depending on the design of the application, there may be other specific error codes that need to be checked for.

Besides checking for return codes that may require specific action by the application, the application should always have a general error routine so that if an unexpected return code is received, the application can alert the user to the problem and provide the user with information about the problem. Unexpected return codes can result from many things including:

- Application programming errors.
- Hardware failure.
- Bugs in either the NDM or the Native ISAM in use.

There are two specific API functions that may be used to extract additional information about an unexpected NDM return code:

31140 NDM_GET_ISAM_ERROR_CODE may be used to extract the native ISAM error code, if any, that occurred during the most recent API function call. This code will be different from the NDM return code and will vary from one native ISAM to another.

31210 NDM_GET_ERROR_DESCRIPTION may be used to extract a text description for either the NDM return code or the native ISAM error code. This information can then be displayed to the user as part of the general error recovery routine.

NOTE: The text description for NDM and Native ISAM error codes is contained in an NDM system file - NDMERR.DAT. These descriptions may be modified (translated to

other languages for example) by use of the Access Error Description File utility. Refer to Chapter 4 for details.

A general error recovery routine may also display information about the environment. The **31160 NDM_GET_CONFIGURATION** function may be used to extract information about the environment including:

- The revision level of NDM.
- The Native ISAM in use, including the revision level of the Native ISAM.

CHAPTER 3

DATA DICTIONARY FILES

3.1 Overview

NDM data dictionary files are files that describe the NDM data files.

These are stored as native ISAM data files, and can therefore be accessed by normal API function calls under control of the application program.

NOTE: The NDM utilities provide facilities for maintaining each type of data dictionary file. Refer to Chapter 4 and Appendix A for more information on the creation and maintenance of these files.

Section 3.2 discusses catalog files.

Section 3.3 discusses data description files.

Section 3.4 discusses key description files.

Section 3.5 discusses relation files.

3.2 Catalog File

The catalog file is used to manage the relationships between data files and their associated data description and key description files. In addition, the catalog file is used to equate platform independent file titles with full path names specific to the system in use.

3.2.1 Advantages of Use

The Catalog File serves three purposes:

- 1) It provides a mechanism for applications to use platform independent file titles to access their application data files. By using the catalog file, the application does not have to be concerned with file naming and file location conventions specific to the platform in use.
- 2) It provides a mechanism by which data files can be associated with their data description and key description files.
- 3) It can be used to store information specific to the native ISAM that is required when a data file is created.

The catalog file typically contains one entry for each data file. This entry contains a file title that is platform independent, the actual physical location of the data file on the platform in use, the file title of the data description file associated with this data file, the file title of the key description file associated with this data file, and the native ISAM specific information.

NOTE: The file titles specified for the data description and key description files themselves point to catalog entries that describe the specific physical location of these files on the platform in use. The catalog entries for the data description and key description files themselves must reference data description and key description file titles. These

file titles are NDM system Data Dictionary files and are discussed further in Chapter 7.

Typically, there is one catalog file per application. This catalog file must contain entries for all NDM System Data Dictionary files and all NDM Support Files. The NDM Utility "Create New Catalog File" will automatically generate proper entries for these files. Refer to Chapter 4 for details.

3.2.2 Layout of Catalog File

The fields of the catalog file are defined in Table 3-1.

Start	Length	Type	Field Name
1	32	A	FILE TITLE
33	72	A	FILE NAME
105	32	A	DATA DESCRIPTION FILE TITLE
137	32	A	KEY DESCRIPTION FILE TITLE
169	128	A	ISAM SPECIFIC
169	1	A	ISAM CODE
170	2	N	PAGE SIZE
172	2	N	PREALLOCATION PAGES
297	184	A	RESERVED FOR NIAKWA EXPANSION

Table 3-1

The field names, in Table 3-1, are defined as follows:

FILE TITLE

A platform independent name for the file that the programmer uses to access the catalog entry.

FILE NAME

Physical location of the data file on the system in use.

DATA DESCRIPTION FILE TITLE

The platform independent name of the data description file for a particular data file. One data description file exists for each NDM data file or class of data files.

KEY DESCRIPTION FILE TITLE

The platform independent name of the key description file for a particular data file. One key description file exists for each NDM data file or class of data files.

ISAM SPECIFIC

Contains information that can be used in the ISAM_SPECIFIC\$ parameter in a call to **31060 NDM_CREATE_FILE**. This field is composed of several sub-fields each of which may or may not apply to the native ISAM in use.

If a sub-field is not applicable under the native ISAM in use, its contents are ignored by the NDM.

ISAM CODE

The code for the native ISAM being used by NDM. Possible values for this, as of NDM revision 1.00, are defined in Table 3-2.

Code	Native ISAM
1	Btrieve
2	C-ISAM
5	SDtrieve

Table 3-2

PAGE SIZE

A page is the smallest unit of storage that the Btrieve moves between system memory and disk. The "pages" are multiples 512 bytes where 512 bytes = 1 page. The maximum number of pages that can be used is 8 or 4096 bytes.

PREALLOCATION PAGES

The number of pages to be preallocated during file creation (Btrieve only). Preallocation can be used to reduce disk fragmentation thereby improving performance.

The indices of a catalog file are defined in Table 3-3.

Index #	Segment #	Unique	Ascending	Field Name
1	0	Y		TITLE
1	1		Y	FILE NAME
2	0	N		NAME
2	1		Y	FILE NAME
3	0	Y		DD TITLE
3	1		Y	DATA DESCRIPTION FILE TITLE
3	2		Y	FILE TITLE

Table 3-3

The contents of a catalog file can be retrieved by calling **31150 NDM_GET_CATALOG_ENTRY**, and are usually only changed by the NDM Utilities. The user is free to change a catalog's contents through normal API calls. Generally, an application calls **31150 NDM_GET_CATALOG_ENTRY** when it needs to open a file. Then it passes the file name to **31340 NDM_OPEN_FILE**. The catalog file is also used by **31070 NDM_CREATE_FILE_FROM_CATALOG**.

3.3 Data Description

The data description file is used to describe the fields and sub-fields of the data record and to define all conversion information for the file. In essence, the data description file is used to define two record layouts for the same file: one for the record as actually stored by the native ISAM, and one for the record as it appears to the Basic-2C application. Each layout contains information about the type and location of each field. One data description file exists for each NDM data file or class of data files. The data description file is used by:

- **31050 NDM_CREATE_CONVERSION_TABLE**
- **31040 NDM_NDM_CREATE_CONVERSION_TABLE_FOR_KEY**
- **31090 NDM_CREATE_KEY_TABLE**
- **31070 NDM_CREATE_FILE_FROM_CATALOG**
- **31290 NDM_GET_RECORD_LENGTH_FROM_DD**

- 31225 NDM_GET_FORMAT_SPEC
- 31223 NDM_GET_FORMAT_SPEC_FOR_KEY.

3.3.1 Data Description File Layout

The fields of the data description file are defined in Table 3-4.

Start	Length	Type	Field Name
1	32	A	FIELD NAME
33	2	N	CONVERSION TABLE NUMBER
35	2	N	B2C FIELD TYPE
37	2	N	B2C FIELD START POSTITION
39	2	N	B2C FIELD TYPE
41	2	N	B2C DECIMAL POSITION
43	2	N	NATIVE FIELD TYPE
45	2	N	NATIVE FIELD START POSITION
47	2	N	NATIVE FIELD LENGTH
49	2	N	NATIVE DECIMAL POSITION
51	128	A	APPLICATION
179	72	A	DESCRIPTION
251	230	A	RESERVED FOR NIAKWA EXPANSION

Table 3-4

The field names in Table 3-4 are defined as follows:

FIELD NAME

The name of the field.

CONVERSION TABLE NUMBER

Used to define multiple conversion tables for one file. This could be used if more than one record type was stored in the same file.

NOTE: In addition, some routines, such as 31290 GET_RECORD_LENGTH_FROM_DD consider only those records that are in conversion table number zero to be a part of the data record.

The fields contained in conversion table zero must define the complete record. However, for higher-numbered conversion tables, only the fields required by the application must be defined. Higher-numbered conversion tables are also useful for defining alternative types of conversion. For example, when converting a key before doing an index look-up, the application may only need to convert a subset of the data record.

B2C FIELD TYPE

This is the field type for the Basic-2C record as used with 31030 NDM_CONVERT. Most field types correspond to the \$PACK field form types. The valid values are defined in Table 3-5.

Type	Class	Min Len	Max Len	Description
1	N	1	255	ASCII free format (10xx)
2	N	1	255	ASCII integer format (2dxx)
3	N	1	255	IBM display format (3dxx)
4	N	1	255	IBM USASCII - 8 format (4dxx)
5	N	1	255	IBM packed decimal format (5dxx)
6	N	1	255	Unsigned packed decimal format (6dxx)
7	A	1	32510	Alpha field (A0xx)
8	N	1	5	Unsigned binary format (Bdxx)
9	N	1	6	Signed binary format (Cdxx)
10	N	8	8	Wang internal numeric format (F0xx)
11	N	8	8	Basic-2C internal numeric format (F1xx)
12	N	4	8	IEEE binary real, H-L format (F2xx)
13	N	4	8	IEEE binary real, L-H format (F3xx)
14	N	4	8	DEC VAX floating point format (F4xx)
15	D	6	6	BASIC-2C \$DATE (YYMMDD)
16	D	3	3	3 byte packed decimal date (YYMMDD)
17	D	3	3	3 byte packed decimal date (MMDDYY)
18	D	4	4	4 byte packed decimal date (YYYYMMDD)
19	D	5	5	ASCII year-days JULIAN date (YYDDD)
20	D	5	5	ASCII days JULIAN date (DDDDD)
21	A	1	32510	Alpha with translation (A0xx)

Table 3-5

NOTE: Type 21 is the same as type 7 except the conversion is first run through a translation table created by the NDM Utilities. Refer to section 4.18 for details. Refer to section 3.3.2 for more information regarding Date Types.

B2C FIELD START POSITION

The start position of the Basic-2C field.

B2C FIELD LENGTH

The length of the Basic-2C field.

B2C DECIMAL POSITION

Records how many digits of the Basic-2C field are to the right of the decimal point for numeric formats.

NATIVE FIELD TYPE

Native fields can either be key or non-key fields. Non-key fields may be any data type supported by the native ISAM. Key fields are normally either native field type 1 for a string or native field type 2 for a two-byte integer. If other field types are used for key fields, then the application must use the key type toolbox feature for extended key types.

Table 3-6 defines the supported native field types, as of revision 1.00 of NDM.

Type	Class	Min Len	Max Len	Description
1	A	1	32511	STRING
2	N	2	2	INTEGER
3	N	4	8	FLOAT
4	A	1	255	LSTRING
5	A	1	32510	ZSTRING
6	D	4	4	DATE
7	A	4	4	TIME
8	N	1	255	Packed Decimal
9	N	1	255	NUMERIC (BTRV)
10	N	2	4	Unsigned binary
11	N	4	4	LONG
12	N	4	8	BFLOAT

Table 3-6

NOTE: Future revisions of NDM will add support for additional native field types. Refer to the Platform Specific Addendum for further details.

NATIVE FIELD START POSITION

The start position of the native ISAM field.

NATIVE FIELD LENGTH

The length of the native ISAM field.

NATIVE DECIMAL POSITION

Records how many digits of the native field are to the right of the decimal point for numeric formats.

APPLICATION

Alphanumeric field available for use by the application.

DESCRIPTION

Narrative description of the purpose of the field.

The indices of a data description file are defined in Table 3-7.

Index #	Segment #	Unique	Ascending	Field Name/Index Name
1	0	Y		NAME
1	1		Y	FIELD NAME
2	0	Y		B2C START
2	1		Y	CONVERSION TABLE NUMBER
2	2		Y	B2C FIELD START POSITION
3	0	Y		NATIVE START
3	1		Y	CONVERSION TABLE NUMBER
3	2		Y	NATIVE FIELD START POSITION
4	0	Y		NATIVE START DESCEN
4	1		Y	CONVERSION TABLE NUMBER
4	2		N	NATIVE FIELD START POSITION

Table 3-7

3.3.2 Date Field Types

Date field types supported by the B2C field type are described in Table 3-8.

Type	Length	Description	Format
15	6	BASIC-2C \$DATE (YYMMDD)	A006
16	3	3 byte packed decimal date (YYMMDD)	6003
17	3	3 byte packed decimal date (YYDDMM)	6003
18	4	4 byte packed decimal date (YYYYMMDD)	6004
19	5	ASCII year-days JULIAN date (YYDDD)	A005
20	5	ASCII days JULIAN date (DDDDD)	A005

Table 3-8

The format column describes the format specification returned by **31225 GET_FORMAT_SPEC** and **31223 GET_FORMAT_SPEC_FOR_KEY**. The following describes each date field type defined in Table 3-8 in more detail:

- 15 The format generated by the Basic-2C \$DATE statement. This is an alpha-numeric value (\$PACK format A006) where each character is one digit of the date in YYMMDD order. The YY portion of the date is the last two digits of the year (i.e., 19YY).
- 16 A six digit number stored as a 3 byte unsigned packed decimal number (\$PACK format 6003). Each digit in the number is one digit of the date in YYMMDD order. The YY portion of the date is the last two digits of the year (i.e., 19YY).
- 17 A six digit number stored as a 3 byte unsigned packed decimal number (\$PACK format 6003). Each digit in the number is one digit of the date in MMDDYY order. The yy portion of the date is the last two digits of the year (i.e., 19YY).
- 18 An eight digit number stored as a 4 byte unsigned packed decimal number (\$PACK format 6004). Each digit in the number is one digit of the date in YYYYMMDD order.
- 19 A 5 digit number stored as an alpha-numeric value (\$PACK format A005) where each character is one digit of the date in YYDDD order. The YY portion of the date is the last two digits of the year (i.e., 19YY). The DDD portion of the date is the number of days from the beginning of the year with the value 1 corresponding to January 1.

- 20 A 5 digit number stored as an alpha-numeric value (\$PACK format A005) where each character is one digit of the date in DDDDD order. The number represented by the DDDDD is the number of days since January 1, 1900 (i.e., a value of 1 corresponds to January 1, 1900).

NOTE: The NDM performs no validly checking on any date field.

3.4 Key Description File

Key description files are used by the following API function calls:

- **31070 NDM_CREATE_FILE_FROM_CATALOG**
- **31040 NDM_CREATE_CONV_TABLE_FOR_KEY**
- **31223 NDM_GET_FORMAT_SPEC_FOR_KEY**
- **31090 NDM_CREATE_KEY_TABLE**

One key description file describes all the indices of a data file or a class of data files. A data file's key description file contains one record for each segment of each index in the data file plus one record to describe the index itself. These two types of entries are distinguished by the SEGMENT NUMBER field. A value of zero for SEGMENT NUMBER is used for index description records. A non-zero SEGMENT NUMBER is used for segment description entries. The fields or sub-fields used as field names in segment description entries must be defined in the data description file and must have a non-zero native field length. For information on how to use the key description file, refer to Chapter 6 of this Programmer's Guide.

3.4.1 Key Description File Layout

The fields of the key description file are defined in Table 3-9.

Start	Length	Type	Field Name
1	32	A	FIELD NAME
33	1	A	ASCENDING
34	1	A	UNIQUE
35	2	N	INDEX NUMBER
37	2	N	SEGMENT NUMBER
39	1	A	KEY COMPRESSION (C-ISAM)
40	55	A	RESERVED FOR NIAKWA EXPANSION

Table 3-9

The field names in Table 3-9 are defined as follows:

FIELD NAME

For segment description entries, the FIELD NAME must be identical to a field name in the associated data description file. Notice the lack of the start position, length, and field types in this record. This information is extracted from the data description file when the key description table is built. For index description entries, the FIELD NAME acts as a description title for the index, but is not used by any API functions.

ASCENDING

Specifies how the segment is ordered:

Y for ascending order

N for descending order

This field is valid for segment description entries only.

UNIQUE

This entry is valid only for index description entries. It defines whether or not duplicate keys are allowed for this index. Y is used for a unique key and N for a non-unique key.

INDEX NUMBER

The number of the index.

SEGMENT NUMBER

Every index must have one or more segments. The segment number defines the order of segments within the index.

KEY COMPRESSION

Contains information in its lowest three bits that describes the types of compression to be performed on key values stored in the file's index. The bits are ignored if the native ISAM does not support this feature. These are defined in Table 3-10.

Bit	Description
0	Leading duplicate characters of a key are compressed.
1	Trailing spaces of a key are removed.
2	Duplicate keys are removed.

Table 3-10

Any or all types of compression may be turned on by setting the corresponding bit to a 1. Use of key compression has no effect on functionality.

The indices of the key description file are defined in Table 3-11.

Index #	Segment #	Unique	Ascending	Field Name
1	0	Y		SEGMENT
1	1		Y	INDEX NUMBER
1	2		Y	SEGMENT NUMBER
2	0	N		NAME
2	1		Y	FIELD NAME
3	0	N		SEGMENT NAME
3	1		Y	SEGMENT NUMBER
3	2		Y	FIELD NAME

Table 3-11

NOTE: If, for a given index number, there is no segment zero record, then duplicate keys are allowed for that index, as if unique were N.

3.5 Relation File

A relation file has one record for each relationship between files in an NDM application. There is typically one relation file per application. Currently, the relation file is used by the NDM Utilities to associate a data description file with a key description file, but this is not used by any API functions. The relation file may be used by applications.

3.5.1 Relation File Format

The data description layout of the relation file is defined in Table 3-12.

Start	Length	Type	Field Name
1	32	A	ACCESSOR FILE TITLE
33	32	A	ACCESSEE FILE TITLE
65	1	A	ONE-TO-ONE
66	32	A	ACCESSOR INDEX
98	32	A	ACCESSEE INDEX
130	95	A	RESERVED FOR NIAKWA EXPANSION

Table 3-12

The filed names in Table 3-12 are defined as follows:

ACCESSOR FILE TITLE

The one in a one-to-many relationship, where the accessee is the many. If the one-to-one flag is set, there is only one accessee record for each accessor record.

NOTE: The use of a relation file requires using a catalog file to look up file titles and, subsequently, data description file names and data description and key description files for both accessor and accessee files to look up key and field names.

ACCESSEE FILE TITLE

The "many" in a one-to-many relationship. If the one-to-one flag is set, there is only one accessee record for each accessor record.

ONE-TO-ONE

A one-to-one relationship is one where there is a direct correlation between one item and another. This value is set to "N" if there is a many-to-one relationship, ie; one accessor record corresponds to many accessee records.

ACCESSOR INDEX

This is used to give the name of the key in the accessee record to SEEK the accessor file.

ACCESSEE INDEX

Refers to the index name given in segment zero of the desired index.

The indices of a relation file are defined in Table 3-13.

Index #	Segment #	Unique	Ascending	Field Name
1	0	N		ACCESSOR FILE
1	1		Y	ACCESSOR FILE TITLE
2	0	N		ACCESSEE FILE
2	1		Y	ACCESSEE FILE TITLE
3	0	N		ACCESSOR INDEX
3	1		Y	ACCESSOR INDEX NAME
4	0	N		ACCESSEE INDEX
4	1		Y	ACCESSEE INDEX NAME

Table 3-13

CHAPTER 4

NDM UTILITIES

4.1 Overview

The following utility programs and menus are supplied by Niakwa for use with the NDM. These utilities are intended to aid the developer with NDM program development in providing a method of easily creating, deleting, viewing, and modifying the NDM support files and application data dictionary files.

Section 4.2 discusses starting the NDM Utilities.

Section 4.3 discusses the Utility Main Menu options.

Section 4.4 discusses the basic key functions in the NDM Utilities.

Section 4.5 discusses the Catalog File Maintenance Utility.

Section 4.6 discusses the Error Description File Maintenance Utility.

Section 4.7 discusses the Field Type File Maintenance Utility.

Section 4.8 discusses the Data Description File Maintenance Utility.

Section 4.9 discusses the Key Description File Maintenance Utility.

Section 4.10 discusses the Access User Data File Utility.

Section 4.11 discusses the Copy User Data File Utility.

Section 4.12 discusses the Export Data Dictionary to IQ Utility.

Section 4.13 discusses the Convert Native Files to Basic-2C Format Utility.

Section 4.14 discusses the Convert Basic-2C Format Files to Native Utility.

Section 4.15 discusses the Set NDMUTIL Options Utility.

Section 4.16 discusses the Install New Catalog File Utility.

Section 4.17 discusses the Display Program Information Utility.

Section 4.18 discusses the Translation File Maintenance Utility.

4.2 Starting the NDM Utilities

The NDM Utilities are typically located in the NDM directory. The supporting programs for the NDM Utilities can be started by following the steps shown below.

- 1) Start the native ISAM if necessary. Refer to the native ISAM's documentation.
- 2) Set the NDM environment variable. Refer to Chapter 2 of the Platform Specific Addendum.
- 3) Change directories into the NDM directory.
- 4) Enter: **NDMUTIL**

NDMUTIL is a batch file that will:

- Invoke the RunTime
- Load the NDM external library
- Load and execute the NDMUTIL boot program

NOTE: The NDM prevents the operator from modifying the data dictionary system catalog file, NDMCAT, which is included with the NDM. This file is the default catalog file the first time the NDM Utilities are started. It is, therefore, necessary to install a new catalog file before creating any new data description or key description files. The installation of a new catalog file copies entries for the existing NDM data dictionary files into the newly created catalog file. Refer to Section 4.4 or 4.16 for more information.

4.3 The Utility Main Menu

When NDMUTIL is executed, the NDM Utilities Main Menu appears as shown in Figure 4-1.

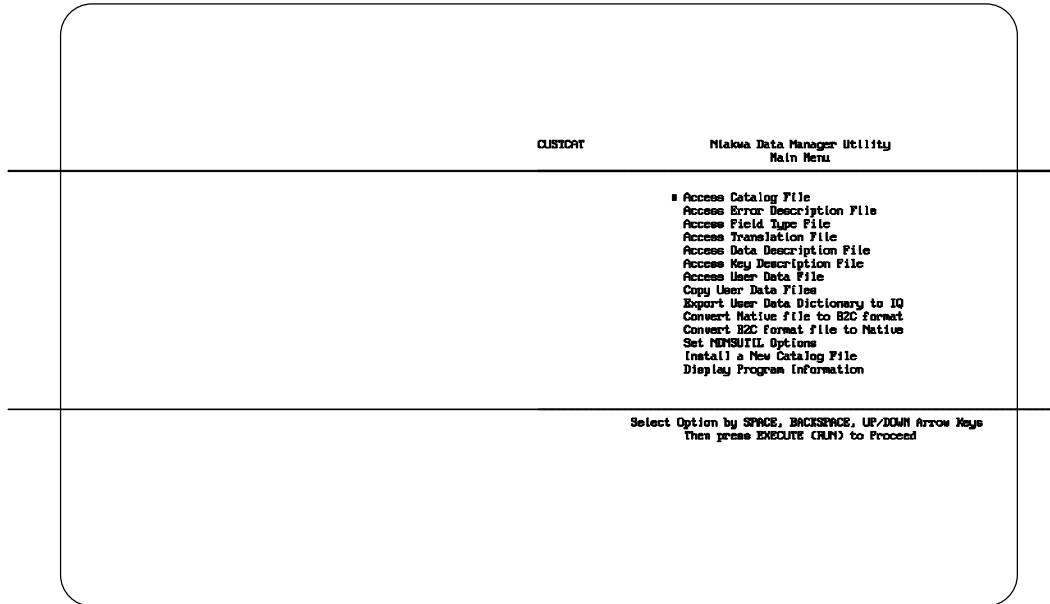


Figure 4 - 1

This Main Menu allows access to all NDM Utility programs.

The following is a brief description of the utility program options listed on the Utilities Main Menu.

Access Catalog File

Displays all file titles and file names in the current catalog file. This also displays of the data description file title, the key description file title, defined page size, ISAM code and preallocation size for a selected file title. This utility also creates new entries and deletes user created entries. The operator can also create native ISAM data files directly from this utility.

Access Error Description File

Displays all NDM return codes and allows the descriptions of these codes to be modified. The primary use of this feature is to allow error messages to be displayed in a language other than English. Developers may also add their own error codes.

Access Field Type File

Displays the Basic-2C and native ISAM supported field types and allows the default field type for native ISAM Fields to be modified.

Access Data Description File

Allows data description files to be created, modified, and deleted.

Access Key Description File

Allows key description files to be created, modified, and deleted.

Access User Data File

Allows the operator to view and modify records in data files associated with the current catalog file.

Copy User Data File

Allows the data contained in an existing data file to be copied a new data file with the necessary data description and key description file changes made. This is necessary if the data description or key description file for an existing data file is modified.

Export User Data Dictionary To IQ

Allows an IQ data dictionary file to be created directly from an NDM data dictionary.

Convert Native Files to Basic-2C Format

Allows the files stored in the native ISAM format to be converted to a Basic-2C diskimage format which provides a method of transferring native ISAM data from one native ISAM to another. Once transferred to another platform, the Convert Basic-2C Format Files to Native Utility is used to convert the files back to the host system's native ISAM format.

Convert Basic-2C Format Files to Native

Allows the data files in a Basic-2C diskimage format to be converted to the native ISAM format. This Utility is used with the Convert Native Files to Basic-2C Format Utility to provide a means of transferring data from one native ISAM to another.

Set NDMUTIL Options

Allows the operator to select a new catalog file to be used by the NDM. Also allows the print address, number of lines per page and print device for all hard copy reports to be specified. The operator can also set the maximum record length allowed. The default and minimum value allowed is 608 bytes.

Install a New Catalog File

Creates a new catalog and relation file. The standard entries from the NDMCAT file are automatically copied to the new catalog file.

Display Program Information

Displays information about the current: version of the native ISAM, NDM, and NDMUTIL program, number of active users, user limit, and value of the NDM environment variable.

Access Translation File

This lets the user add, modify and delete records in the translate file. The records are used by **31425 NDM_SET_TRANSLATION_TABLE** for alpha field type translations from Basic-2C to native ISAM format and back to Basic-2C.

4.4 NDM Utility Operation

Table 4-1 outlines the key functions available throughout the NDM Utilities.

Instructions are generally provided at the bottom of the various utility screens.

If after selecting an option a cursor does not appear, then the item selected cannot be modified.

NOTE: No NDM data dictionary system file may be modified.

4.4.1 Main View Screen Operation

Once a utility is selected, the Main View Screen for that utility appears. The keys defined in Table 4-1 then become active.

Key	Function
PgUp	View the previous page of information
PgDn	View the next page of information.
Up Arrow	Highlight the previous item
Down Arrow	Highlight the next item
End (Cancel)	Exit current view screen
Execute	Select the highlighted item
Insert	Insert a new item
Delete	Delete the current item.
P	Print the file being viewed
R	Reorganizes data description file (Access Data Description File, only)
C	Create user data file. (Access Catalog File, only)
Tab	Allows selection of a new index and starting value by which to order the current list of items.

Table 4-1

4.5 Catalog File Maintenance

The Catalog File Maintenance Utility displays the file titles and file names used in the current catalog file. For a selected file title, it displays the data description file title, key description file title, native ISAM code, defined page size, and pre-allocation size.

4.5.1 Features

This utility:

- Lists all the file titles and file names in the current catalog file.
- Allows any file title in the current catalog file to be selected.
- Displays the data description, key description, native ISAM code, pre-allocation size, and page size of the selected title.
- Allows new entries in the catalog file (must be user data files) to be created.
- Allows user data file entries to be deleted.
- Allows user data files to be created or modified.
- Allows file names for the data description and key description entries to be modified.

4.5.2 General Use

Select this utility from the Utilities Main Menu. A list of the file titles and file names in the current catalog file appears on the Utilities Main View Screen.

To select a file title, use the Arrow keys to highlight the name and press Execute.

Once a selection is made, the utility displays the associated data description file title, key description file title, page size and pre-allocation size of the selected file title as shown in Figure 4-2.

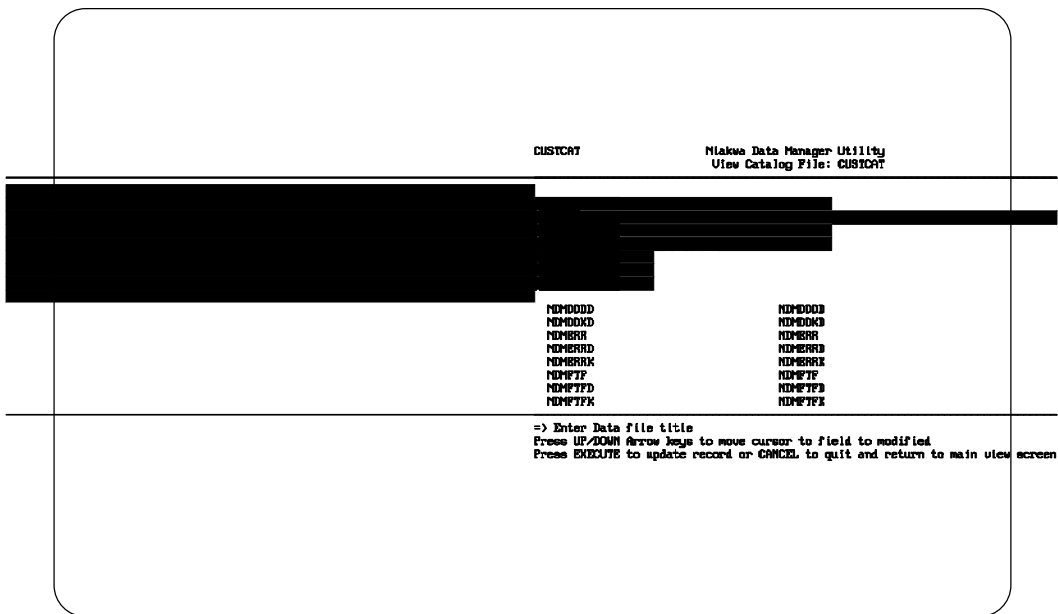


Figure 4 - 2

To modify a catalog file entry, make the desired changes and press Execute to update the entry or Cancel to exit without updating.

NOTE: Only entries for user data files may be modified. File names for data description and key description files may not be modified.

The utilities that create the data description and key description files will automatically create the appropriate entries in the current catalog file. It is only necessary to create a catalog entry for the user data file.

To create a new entry, press Insert. The window shown in Figure 4-3 then appears.

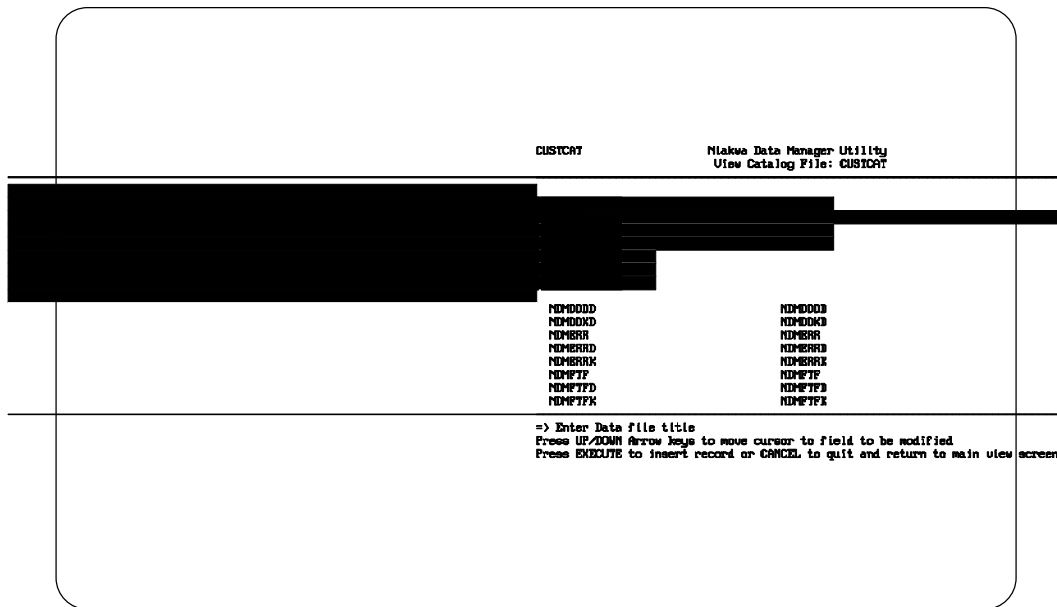


Figure 4 - 3

Enter a name for the File Title (this is a mnemonic name for the data file). Next enter the physical location of the file. A full pathname should be specified. Typically, no extension should be specified. For the data description and key description file entries, a window of valid data description and key description file titles appears. Press the Tab key to allow selection of the window entries. Select the file by using the Arrow keys and press Execute.

Once the data description and key description files have been selected, the last three fields must be entered.

The ISAM code is a code number associated with the native ISAM in use. A window appears with a list of the supported ISAMs. Select the native ISAM for the host system.

The Page Size is the smallest unit of storage that a native ISAM moves between main memory and disk. This must be a multiple of 512.

The preallocation field tells the native ISAM how many pages should be preallocated when creating the file. For example, if the page size is 1024 and the preallocation number is 4, a file will be created that has an initial size of 4096 bytes.

NOTE: ISAM code, page size, and preallocation size, are native ISAM dependent fields. Please refer to the native ISAM manual for more information on these fields.

Pressing Execute creates the new entry. The utility remains in INSERT mode. Additional catalog entries can be made as above or press Cancel to return to the Main View Screen.

To create the actual data file, select the data file with the arrow keys and press C to create the file. A window appears, confirming the choice. Press Enter to create the file or enter "N" to not create the data file.

NOTE: If the data file already exists, a message appears indicating this. To recreate a file, it is necessary to delete it first by using the 31100 NDM_DELETE_FILE function in a user written program.

To delete a data file entry, select the data file entry with the arrow keys and press D to delete the entry. A window appears confirming the choice.

NOTE: Only the data file entry is deleted, the data file itself is not deleted.

To return to the Utilities Main Menu, press Cancel from the Main View Screen.

4.6 Error Description File Maintenance

The Error Description File Maintenance Utility displays all NDM and native ISAM error code descriptions. The developer can modify these descriptions to meet the special needs of the application. For example, the descriptions can be modified to correspond to the native language of the end user. The developer can also add additional error codes and descriptions.

For a detailed discussion on these codes, refer to Chapter 11 of this Programmer's Guide.

4.6.1 Features

This utility:

- Lists all the error codes and text descriptions of the NDM and the native ISAM.
- Allows the use of Page Up and Page Down keys to view the various error codes.
- Allows the operator to edit the error descriptions.
- Allows the operator to add new error codes and descriptions.

4.6.2 General Use

Select this utility from the Utilities Main Menu. The NDM error codes are displayed one screen at a time. Use Page Down to display the next screen or Page Up to display the previous screen. An example error code screen is shown in Figure 4-4.

```

CUSTDAI                               Niakwa Data Manager Utility
                                       View NDM Error File
-----
TYPE CODE      DESCRIPTION
# 0 0 No error - the function completed successfully.
0 1 End of file was encountered.
0 2 Beginning of file was encountered.
0 3 File contains no records.
0 4 I/O error.
0 5 File is not open.
0 6 Key value was not found in file.
0 7 Duplicate keys are not allowed.
0 8 No index. The "current index" is undefined.
0 9 No position. The "current record" is undefined.
0 10 Invalid file name.
0 11 File was not found.
0 12 Error while closing file.
0 13 Disk is full.

Press PREVIOUS to view file, UP/DOWN Arrow to select record, TAB to select index
EXECUTE/INSERT/DELETE keys to view-modify/insert/delete selected record
Press F to print file, CANCEL key to return to main menu
    
```

Figure 4 - 4

To edit an error description, use the Arrow keys to select a description and press Execute. A window appears as shown in Figure 4-5.

```

CUSTDAI                               Niakwa Data Manager Utility
                                       View NDM Error File
-----
TYPE CODE      DESCRIPTION
# 0 0 No error - the function completed successfully.
0 1 End of file was encountered.
0 2 Beginning of file was encountered.
0 3 File contains no records.
0 4 I/O error.
0 5 File is not open.
0 6 Key value was not found in file.
0 7 Duplicate keys are not allowed.
0 8 No index. The "current index" is undefined.
0 9 No position. The "current record" is undefined.
0 10 Invalid file name.
0 11 File was not found.
0 12 Error while closing file.
0 13 Disk is full.

=> Enter error description
Press UP/DOWN Arrow keys to move cursor to field to be modified
Press EXECUTE to update record or CANCEL to quit and return to main view screen
    
```

Figure 4 - 5

The utility will only allow the operator to modify the description of an NDM defined error code. The return type and return code numbers cannot be modified. Enter the desired changes to the description field and press Execute. The utility then displays the main view screen again.

To add a new error description, press Insert. A window appears as shown in Figure 4-6.

```

                                CUSTOM          Ndamw Data Manager Utility
                                View NDM Error File
-----
                                TYPE CODE      DESCRIPTION
                                [REDACTED]
                                0  5 File is not open.
                                0  6 Key value was not found in file.
                                0  7 Duplicate keys are not allowed.
                                0  8 No index. The "current index" is undefined.
                                0  9 No position. The "current record" is undefined.
                                0 10 Invalid file name.
                                0 11 File was not found.
                                0 12 Error while closing file.
                                0 13 Disk is full.

=> Enter error type
Press UP/DOWN Arrow keys to move cursor to field to be modified
Press EXECUTE to insert record or CANCEL to quit and return to main view screen

```

Figure 4 - 6

Enter the appropriate information in the fields provided, using the Arrow or Enter keys to move between fields. Press Execute to save/update the return code file.

NOTE: Any new error code descriptions added by the developer must be of type 100 or greater. This utility will not allow entry of any new codes for types 0 through 99.

When finished with an entry the utility will remain in INSERT mode, for additional error code entries. Once all entries have been completed, press Cancel to return to the Utilities Main Menu.

To delete a user created error code and description, select the error code record by using the Arrow Keys. To delete the error code, press Delete. A window appears prompting the operator to enter a "Y" to confirm the deletion of the error description. Press Enter to confirm, or enter "N" and press Enter to prevent the deletion of the error description.

To return to the Utilities Main Menu, press Cancel.

4.7 Field Type File Maintenance

The Field Type File Maintenance Utility lists the allowed Basic-2C and native ISAM field types. Refer to Chapter 2, Chapter 3, and the Platform Specific Addendum for more information on the NDM and native ISAM supported field types.

4.7.1 Features

This utility:

- Displays the supported field types for Basic-2C and the native ISAM.
- Allows the use of Page Up and Page Down to list all field types.
- Allows the default field type for native ISAM fields to be modified.

Figure 4-7 displays a sample screen used by this utility.

CUSTCAT		Makwa Data Manager Utility View NDM Field Type File							
TYPE	LEN	CLASS	MIN	LEN	MAX	LEN	DEC	DEFP?	DESCRIPTION
# 1	B	N	1		255	1		N	ASCII FREE FORMAT
2	B	N	1		255	1		N	ASCII INTEGER FORMAT
3	B	N	1		255	1		N	IBM DISPLAY FORMAT
4	B	N	1		255	1		N	IBM USASCII-8 FORMAT
5	B	N	1		255	1		N	IBM PACKED DECIMAL FORMAT
6	B	N	1		255	1		N	UNSIGNED PACKED DECIMAL FORMAT
7	B	A	1		32510	1		N	ALPHA FIELD
8	B	N	1		5	1		N	UNSIGNED BINARY FORMAT
9	B	N	1		5	1		N	SIGNED BINARY FORMAT
10	B	N	8		8	0		N	WANG INTERNAL NUMERIC FORMAT
11	B	N	8		8	0		N	BASIC-ZE INTERNAL NUMERIC FORMAT
12	B	N	4		8	4		N	IEEE BINARY REAL, H-L FORMAT
13	B	N	4		8	4		N	IEEE BINARY REAL, L-H FORMAT
14	B	N	4		8	4		N	DEC VAX FLOATING POINT FORMAT

Press FRENCH to view file, UP/DOWN Arrow to select record, TAB to select index
 EXECUTE key to view/modify selected record
 Press F to print file, CANCEL key to return to main menu

Figure 4 - 7

NOTE: For details on the format of each native ISAM field type, refer to the native ISAM documentation.

4.7.2 General Use

Select this utility from the Utilities Main Menu. The supported Basic-2C field types are displayed. Pressing the Page Down key displays more Basic-2C field types followed by the supported native ISAM field types. Page Up will return the operator back to previous pages. The B/N specifications identifies whether the supported field type is a Basic-2C field type or a Native ISAM field type (B for Basic-2C and N for Native).

To modify the default field type, select the Native field type with the Arrow keys and press Execute.

To modify the DEFAULT field, enter either "Y" or "N" and press Execute.

NOTE: Only one string type, one numeric type, and one date type should be selected as the default field at any one time.

To return to the Utilities Main Menu press Cancel.

4.8 Data Description File Maintenance

The Data Description File Maintenance Utility allows data description files to be modified, created, copied, and deleted. The data description file describes each field in the data file. It also defines all conversion information necessary to convert records between Basic-2C format and the native ISAM format.

4.8.1 Features

This utility:

- Lists all the data dictionaries in the catalog file.
- Displays the field definitions of the selected data description files.
- Allows new data description files to be created.
- Allows existing data description files to be copied to a new file (this allows the developer to modify a data description while still keeping the old file in its original form).
- Allows existing data description files to be modified.
- Allows user created data description files to be deleted.
- Allows the data field records be reorganized.

4.8.2 General Use

Select this utility from the Utilities Main Menu. A window appears the names of the existing data description files in the current catalog file. The operator has the choice of creating a new data description file or viewing/modifying an existing data description as shown in Figure 4-8.

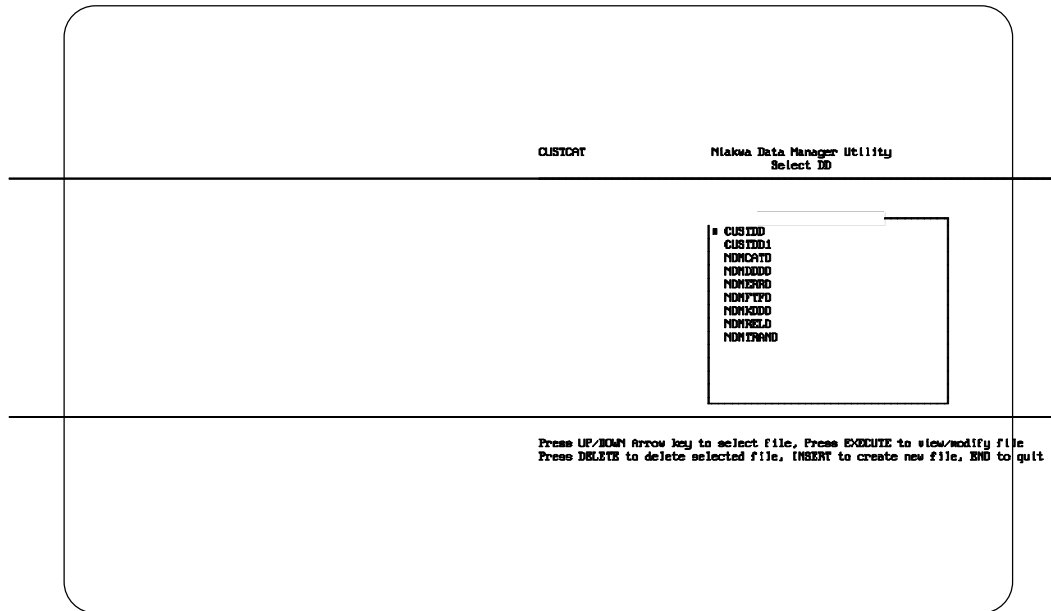


Figure 4 - 8

To create a new data description file refer to Section 4.8.3

To view/modify a data description file refer to Section 4.8.4

To reorganize a data description file refer to Section 4.8.5

To delete a data description file, use the arrow keys to select the file and press Delete. A window appears asking for confirmation of the deletion. The default answer is "N". Type "Y" and press Enter to delete or Cancel to prevent the deletion of the file.

4.8.3 Creating A New Data Description

To create a new data description file, press Insert while the display window of the data description file names is still present (refer to Section 4.8.2).

A screen appears requiring the operator to enter the names of the file title and file name for the new data description file. Enter the file's appropriate fields and press Execute.

Next, a window appears, prompting the operator to select a data description file to be used as a source file for the new data description file. This allows the user to update an existing data description file, by copying the existing data description file records into a new file and then modifying the new file.

Select a source file and press Execute. If no source file is necessary, press Cancel to create a new data description file.

If a source file was selected, a screen displaying the fields of the data description file appears. If a completely new data description file is being created, a data entry screen appears, similar to the one shown in Figure 4-9.

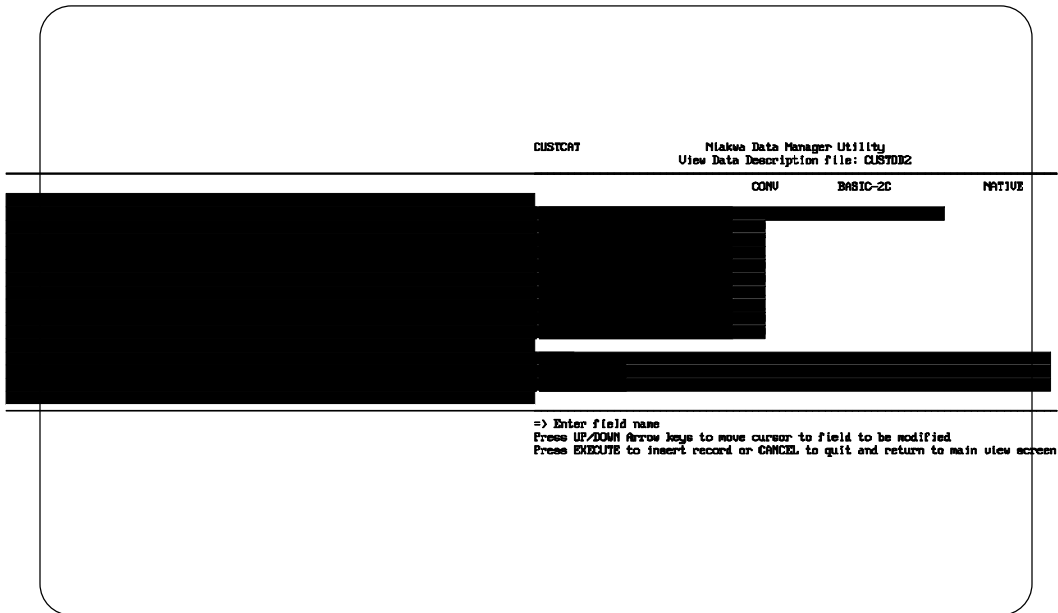


Figure 4 - 9

The above entry fields will describe one field of the data description record being created. Enter this information one field at a time. Some entry fields, such as field type will display a window. The operator may either enter one of the field types that appears or may use the Tab key to move to the selection window.

Once in the selection window, use the Arrow Keys, PREV, or NEXT to highlight the desired entry and press Execute to select. Press Cancel to exit the selection window with no changes.

NOTE: When in INSERT Mode, the utility will provide reasonable default values for Native Field Type, Native Start Position, and Native Length based on the Basic-2C information entered. These defaults may be modified.

After all required information for each field is entered, press Execute to continue. The utility will remain in INSERT mode, allowing the operator to enter additional fields.

The above entry fields must be completed for each additional field of the data file being described. Refer to Chapter 3 of this Programmer's Guide for more detailed information on the entry fields.

When finished entering all field descriptions in the data description file, press Cancel to return to the Main View Screen. A description of all new fields are displayed as in the example shown in Figure 4-10.

CUSTCAT		Niakwa Data Manager Utility View Data Description file: CUSTDD								
FIELD NAME	CONV	TABLE	BASIC-2C			NATIVE				
			TYPE	START	LEN	DEC	TYPE	START	LEN	DEC
■ CUSTOMER NUMBER	8	7	1	5	0		1	1	5	0
NAME	8	7	6	28	0		1	6	28	0
PHONE	8	7	26	13	0		1	26	13	0
YTD SALES\$	8	5	39	5	0		9	39	5	0
NUMBER OF SALES	8	5	44	2	0		2	44	2	0
CITY	8	7	46	28	0		1	46	28	0
NAME-1	1	7	6	5	0		1	6	5	0
AREA-CODE	2	7	27	3	0		1	27	3	0

Press FREQ/TEXT to view file, UP/DOWN Arrow to select record, TAB to select index
 EXECUTE/INSERT/DELETE keys to view/modify/insert/delete selected record
 Press F to print file, R to reorg file, Cancel key to return to main menu

Figure 4 - 10

If necessary, fields may be modified by moving the cursor to the desired field and pressing Execute.

To return to the Utilities Main Menu press Cancel.

4.8.4 Modifying/Viewing a Data Description File

To modify or view an existing data description, select a data description file from the display window described in Section 4.8.2.

NOTE: Niakwa strongly encourages developers to make modifications only to a working copy of the data description file. This can easily be done by creating a new data description file and specifying the file to be modified as 'source'. It is absolutely essential that the original data description file be retained if an actual data file based on it exists. Failure to retain the original data description file may result in fatal errors when attempting to access the user data file.

NOTE: Only actual user data description files can be modified. Data description system files can only be viewed.

A listing showing all the field names in the data description file appears as shown in the example screen in Figure 4-11.

CUSTCAT		Niakwa Data Manager Utility View Data Description file: CUSTDD									
FIELD NAME	CONV TABLE	BASIC-2C				NATURE		LEN		DEC	
		TYPE	START	LEN	DEC	TYPE	START	LEN	DEC		
■ CUSTOMER NUMBER	0	7	1	5	0	1	1	5	0		
NAME	0	7	6	20	0	1	6	20	0		
PHONE	0	7	26	13	0	1	26	13	0		
VTD SALES\$	0	5	39	5	0	9	39	5	0		
NUMBER OF SALES	0	6	44	2	0	2	44	2	0		
CITY	0	7	45	20	0	1	45	20	0		
NAME-1	1	7	6	5	0	1	6	5	0		
AREA-CODE	2	7	27	3	0	1	27	3	0		

Press FREQ/NEXT to view file, UP/DOWN Arrow to select record, TAB to select index
 EXECUTE/INSERT/DELETE keys to view/modify/insert/delete selected record
 Press P to print file, R to reorg file, Cancel key to return to main menu

Figure 4 - 11

To modify a field, select a field to modify and press Execute. The information fields of the data field can now be modified by typing over the existing information as in the example screen in Figure 4-12.

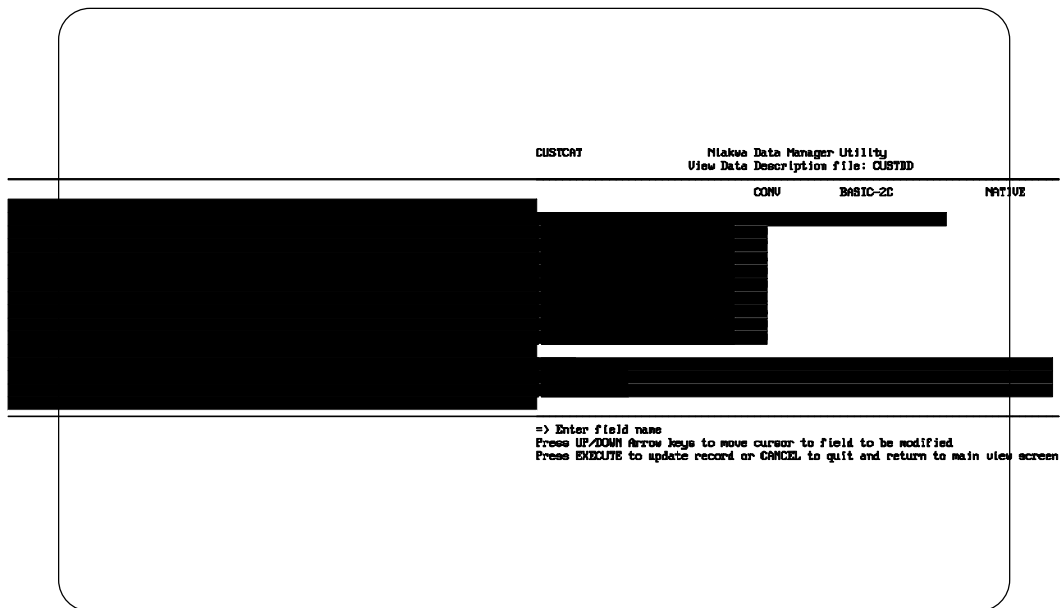


Figure 4 - 12

After completing any modifications, press Execute to return to the Main View Screen.

The Cancel key may be pressed to return to the field list without modifying the entry.

The operator may now choose another data field to modify/view or return to the Utilities Main Menu by pressing Cancel.

Data description entries may be inserted or deleted from the modify/view screen.

4.8.5 Data Description Reorganization

It may be necessary to reorganize the records contained within a data description file after modifications, additions or deletions to the data description have caused overlapping fields or "holes" within the record layout. Pressing R while at the view data description file screen will perform this reorganization.

When R is pressed, this utility will prompt the operator for whether the data description file should be reorganized by Basic-2C or native field start positions. Once the operator has made a choice, the utility reads through all the data description records and changes the start position of the corresponding fields to close any "holes" (created by deleted fields) or fix any overlapping fields for conversion table 0.

For all other conversion tables, the start positions are changed to keep the fields starting in the same start position relative to the field in conversion 0. This will keep sub-fields of fields in the correct position.

4.9 Key Description File Maintenance

The Key Description File Maintenance Utility allows key description files to be created, copied, deleted, and definition of the indices within a native ISAM data file used with the NDM.

4.9.1 Features

This utility:

- Lists all key description files in the catalog file.
- Allows new key description files to be created.
- Allows existing key description file information to be copied into a newly created key description file.
- Allows new index entries to be created.
- Allows existing index entries to be modified or deleted.

4.9.2 General Use

Select this utility from the Utilities Main Menu. A screen appears with the names of all key description files in the current catalog file. Refer to the example shown in Figure 4-13.

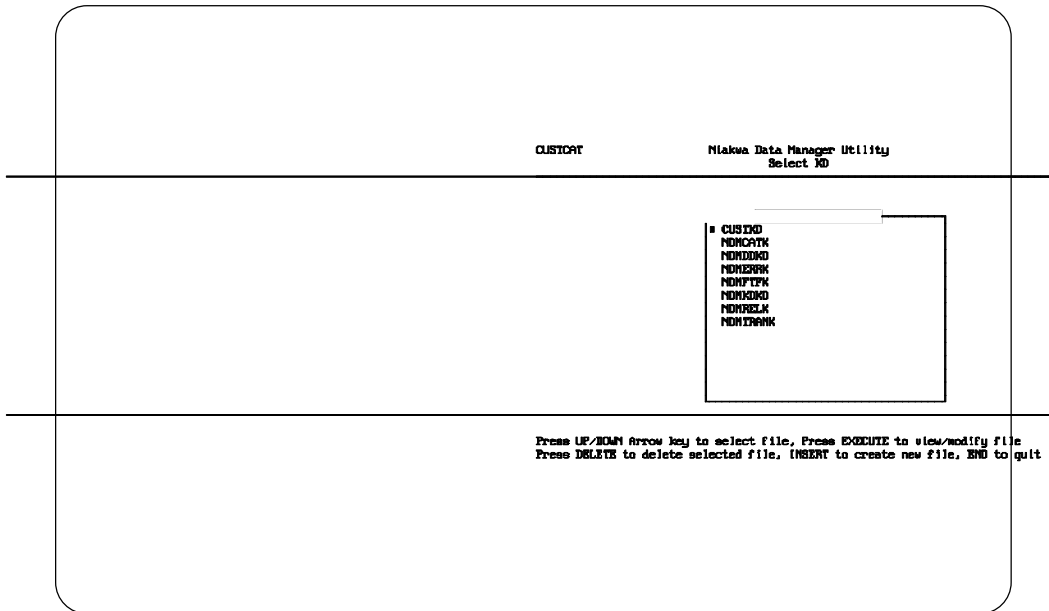


Figure 4 - 13

The operator has the option of creating a new key description file or viewing/modifying an existing key description file.

To create a new key description file refer to Section 4.9.3. To view/modify a key description file refer to Section 4.9.4. To delete a key description file use the Arrow keys to select the file to delete and press Enter. A window appears for confirmation of the choice. Press Enter to confirm or Cancel to cancel.

4.9.3 Creating a New Key Description File

To create a new key description file, press Insert while the window of key description file names is still visible (refer to Section 4.9.2).

A screen appears requiring the operator to enter the file title and file name for new key description file. Enter the required file names in the appropriate fields and press Execute.

Next, a window appears, prompting the operator to select a key description file to be used as a source file for the new key description file. This feature makes updating existing key description files easier, by permitting the copying of existing key description file descriptions into a new file and then modifying this file. This also allows the developer to modify a key description file while keeping the original intact.

Select a source file and press Execute. If no source file is necessary, press Cancel to create a new key description file.

The operator is then prompted to enter the file title of the related data description file for this key description file. A window appears displaying a list of all data description file titles. Select the related data description file and press Execute. The screen in Figure 4-14 then appears as shown below.

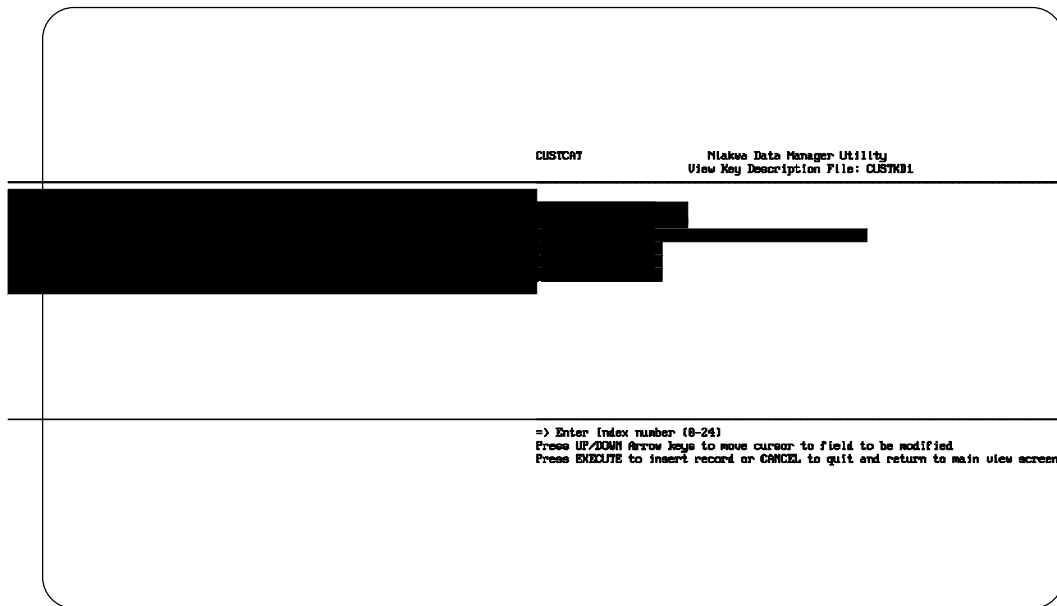


Figure 4 - 14

The above entry fields describe one segment of one index of the key description file for the related data description file.

Enter the information for segment 0 of the first index that is being created and press Execute.

Refer to Chapter 3 of this Programmer's Guide for more information on these entry fields.

NOTE: For each index created, at least 2 segments are required. The first, segment 0, provides a mnemonic name for the index. When entering information for segment 0, the operator is not prompted for the Ascending field, only for the Unique field. Uniqueness is only described in segment 0. Segment 1 is the first actual segment.

Once the operator enters a number other than 0 for a segment number, the field name Index Name changes to Field Name. When the operator is entering information for the Field Name, a window appears with the field names from the related data description. The operator should then select the appropriate field name for this segment from this window.

NOTE: When entering information on a non-zero segment, the utility allows the operator to specify the Ascending field.

After entering the information for a segment press Execute to save. The utility remains in INSERT mode and allows the entry of additional segments and indices.

When all index information has been added press Cancel. The Main View Screen then appears, similar to the one shown in Figure 4-15, listing the information for each index and segment entered.

CUSTCAT		Makwa Data Manager Utility View Key Description File: CUSTKD				
FIELD NAME	ASC?	UNIQUE	INDEX	SEGMENT	COMP?	
# CUST #		V	1	0		
CUSTOMER NUMBER	V		1	1		
SORT NAME		N	2	0		
NAME-1	V		2	1		
SALES BY AREA		N	3	0		
AREA-CODE	V		3	1		
VTB SALES\$	N		3	2		

Press FREQ/NEXT to view file, UP/DOWN Arrow to select record, TAB to select index
 EXECUTE/INSERT/DELETE keys to view/modify/insert/delete selected record
 Press F to print file, CANCEL key to return to main menu

Figure 4 - 15

Pressing Cancel returns the operator to the Utilities Main Menu.

4.9.4 Modifying/Viewing a Key Description File

To modify or view a key description file, select a key description file from the display window described in Section 4.9.2 and shown in Figure 4-16.

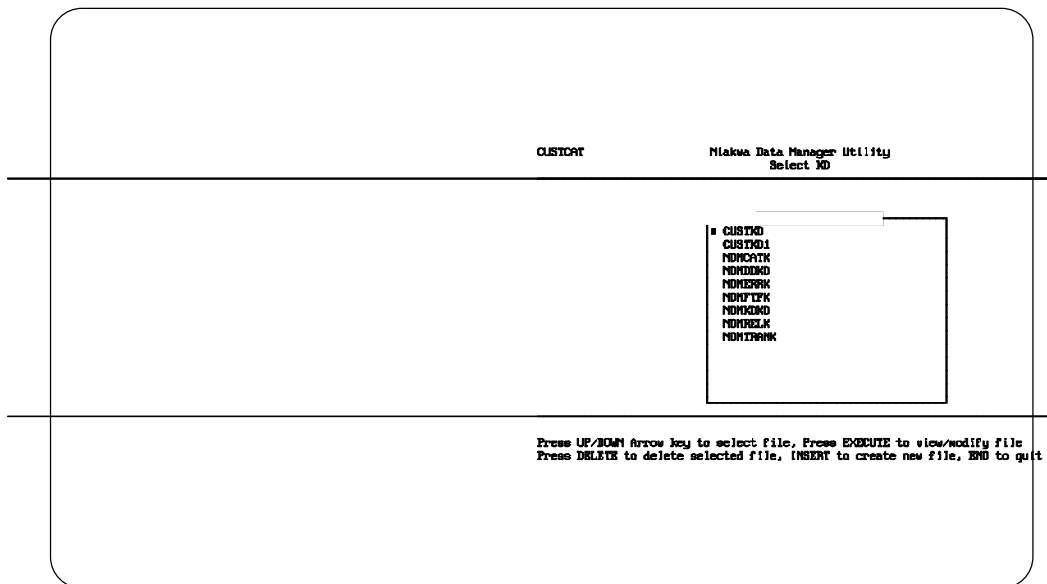


Figure 4 - 16

Select the key description file and press Execute.

NOTE: Only key description files for user data can be modified. Data dictionary system key description files may only be viewed.

NOTE: Niakwa strongly encourages developers to make modifications only to a working copy of the key description file. This can easily be done by creating a new key description file and specifying the file to be modified as "source". It is absolutely essential that the original key description file be retained if an actual data file based on it exists. Failure to retain the original key description file may result in fatal errors when attempting to access the user data file.

A list is then shown displaying all created indices in the key description file as shown in Figure 4-17.

CUSTCAT		Niakwa Data Manager Utility View Key Description File: CUSTKD			
FIELD NAME	ASC?	UNIQUE	INDEX	SEGMENT	COMP?
# CUST #		V	V	1	0
CUSTOMER NUMBER	V			1	1
SORT NAME		N	N	2	0
NAME-1	V			2	1
SALES BY AREA		N	N	3	0
AREA-CODE	V			3	1
VTB SALES\$	N			3	2

Press F8U/NEXT to view file, UP/DOWN Arrow to select record, TAB to select index
 EXECUTE/INSERT/DELETE keys to view/modify/insert/delete selected record
 Press F to print file, CANCEL key to return to main menu

Figure 4 - 17

To select a segment, press Execute. Make any necessary modifications and then press Execute to update and return to the Main View Screen.

To delete a segment press Delete.

To return to the Utilities Main Menu, press Cancel.

4.10 Access User Data Files

The Access User Data File Utility allows the operator to view, modify and create records for a specified data file (the data file can be created by use of the Access Catalog File Utility. Refer to Section 4.5).



No provisions are made to verify the data entered.

4.10.1 Features

This utility:

- Lists all the user data files associated with the current data description.
- Allows the user to select a data file.
- Displays the records of the selected file.
- Allows the operator to select the Index and starting values by which to order the list displayed.
- Allows the operator to select a specified record to view, modify, or delete.
- Allows the insertion of new records.

NOTE: If a field is longer than 44 bytes, it is treated as several distinct fields of length 44 or less. This prevents wrap around occurring between two lines of the same field when using the Insert or Delete keys.

4.10.2 General Use

Select this utility from the Utility Main Menu. A window appears with a list of the associated data files for the current catalog file as shown in Figure 4-18.

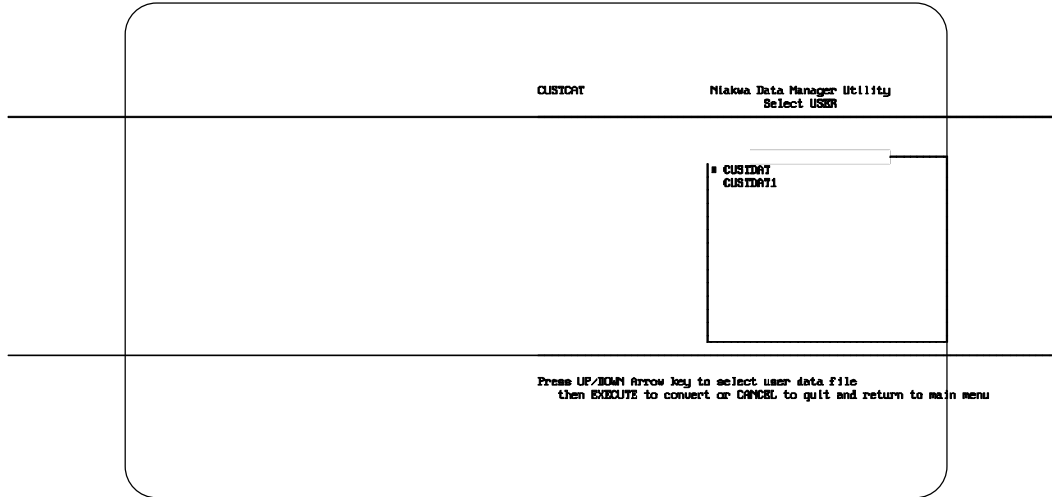


Figure 4 - 18

To select a data file, use the Arrow Keys to select the file name and press Execute.

Once a selection is made, the utility displays the records of the selected file as shown in Figure 4-19.

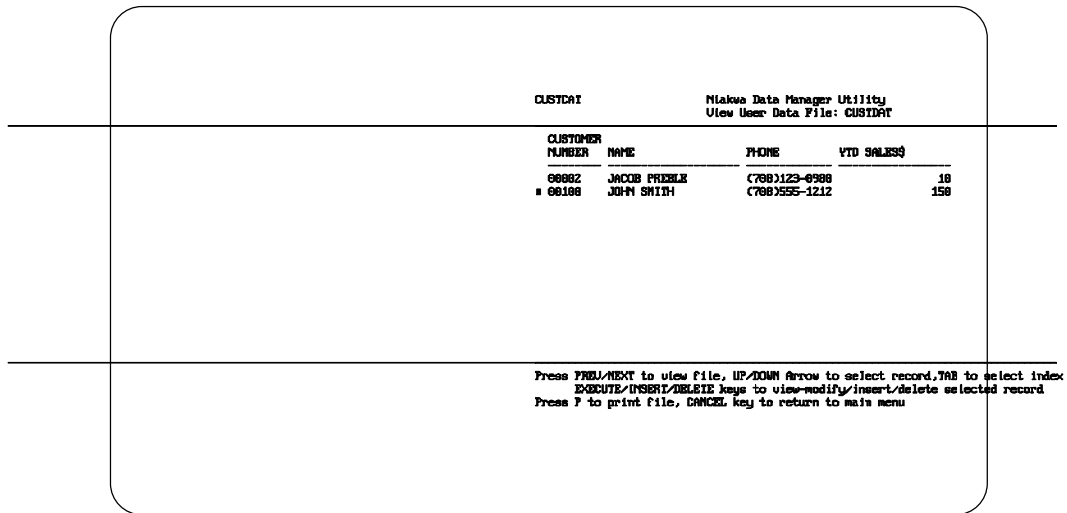


Figure 4 - 19

NOTE: If the selected file does not exist, the operator is asked if the file should be created.

To insert a new record, press Insert. A screen similar to the one in Figure 4-20 is then displayed.

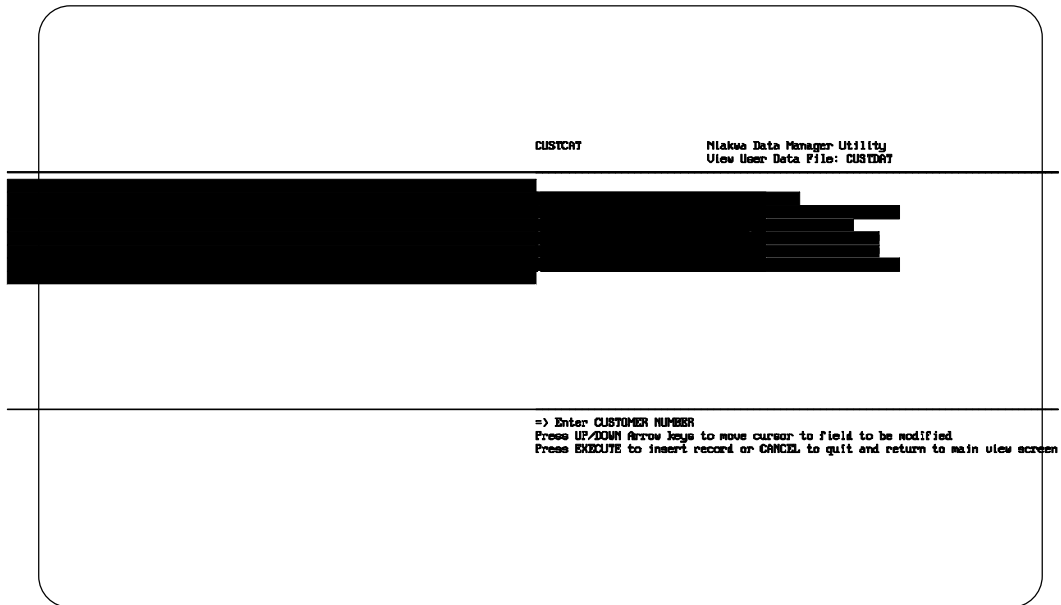


Figure 4 - 20

Enter the appropriate data in the fields of the record and press Execute to save. The utility stays in INSERT mode and displays a new data entry screen.

Enter any additional records in the same manner as above or press Cancel to return to the Main View Screen. A screen similar to the one in Figure 4-21 is then displayed.

CUSTDAT		Niakwa Data Manager Utility View User Data File: CUSTDAT		
CUSTOMER NUMBER	NAME	PHONE	YTD SALES\$	
60002	JACOB PREELE	(708)123-0988	10	
60100	JOHN SMITH	(708)555-1212	150	

Press FREQ/NEXT to view file, UP/DOWN Arrow to select record, TAB to select index
 EXECUTE/INSERT/DELETE keys to view/modify/insert/delete selected record
 Press F to print file, CANCEL key to return to main menu

Figure 4 - 21

To select a new index and starting values for ordering the display, press Tab. A window appears showing the indices of the file. An index can be selected by highlighting the name of the desired index and pressing Execute. A new window will appear that allows the operator to enter the starting values to display. Entering the desired values and press Execute.

To make modifications to a record, select the record with the use of the Arrow keys and press Execute. A data record edit screen then appears similar to the one in Figure 4-22.

CUSTCAT		Niakwa Data Manager Utility View Key Description File: CUSTMD				
FIELD NAME	ASC?	UNIQUE	INDEX	SEGMENT	COMP?	
# CUST #		V	V	1	0	
CUSTOMER NUMBER		V		1	1	
SORT NAME			N	2	0	
NAME-1		V		2	1	
SALES BY AREA			N	3	0	
AREA-CODE		V		3	1	
YTD SALES\$		N		3	2	

Press FREQ/NEXT to view file, UP/DOWN Arrow to select record, TAB to select index
 EXECUTE/INSERT/DELETE keys to view-modify/insert/delete selected record
 Press F to print file, CANCEL key to return to main menu

Figure 4 - 22

Make any necessary modifications using the Arrow keys or the Enter key to move between fields. Press Execute to update the record. To return to the Utilities Main View Screen, press Cancel and a screen similar to the one in Figure 4-23 appears.

CUSTCAT		Niakwa Data Manager Utility View User Data File: CUSTDAT		
CUSTOMER NUMBER	NAME	PHONE	YTD SALES\$	
# 00002	JACOB PREBLE	(708)123-0988	10	
# 00100	JOHN SMITH	(708)555-1212	150	

Press FREQ/NEXT to view file, UP/DOWN Arrow to select record, TAB to select index
 EXECUTE/INSERT/DELETE keys to view-modify/insert/delete selected record
 Press F to print file, CANCEL key to return to main menu

Figure 4 - 23

To delete a record from this screen, use the Arrow keys to select a record and press Execute to delete.

To return to the Utilities Main Menu, press Cancel.

4.11 Copy User Data File

The Copy User Data File Utility allows the developer to make a copy of the data contained in a data file after the data description or key description files have been modified. This allows existing data files to be easily modified for any changes made to the data description or key description files (i.e. a data field is removed or added).

4.11.1 Features

This utility:

- Allows the developer to copy data from an original data file to a new data file while reorganizing the data based on modifications made to the data description or key description files.

4.11.2 General Use

Select this utility from the Utilities Main Menu. Two windows appear as shown in Figure 4-24.

CUSTCAT Niakwa Data Manager Utility
COPY USER DATA FILE

=> Enter source user data file OR Press TAB to select from menu
Press UP/DOWN Arrow keys to move cursor to field to be modified
Press EXECUTE to start copy or CANCEL to quit and return to main menu

Figure 4 - 24

The top window displays prompts related to the information that this utility needs to perform its function. The first two fields, Source File and Destination File must be entered by the operator.

The bottom window displays the data files in the current catalog file that can be used as valid responses for these files. To enter this second window, press Tab while the cursor is positioned at either the Source File field or the Destination File field. Once in this window, the operator can select a data file by using the Arrow keys and pressing Execute.

Once both Source File and Destination File have been selected, pressing Execute starts the copy process. The utility then displays the status of the copy as shown in Figure 4-25.

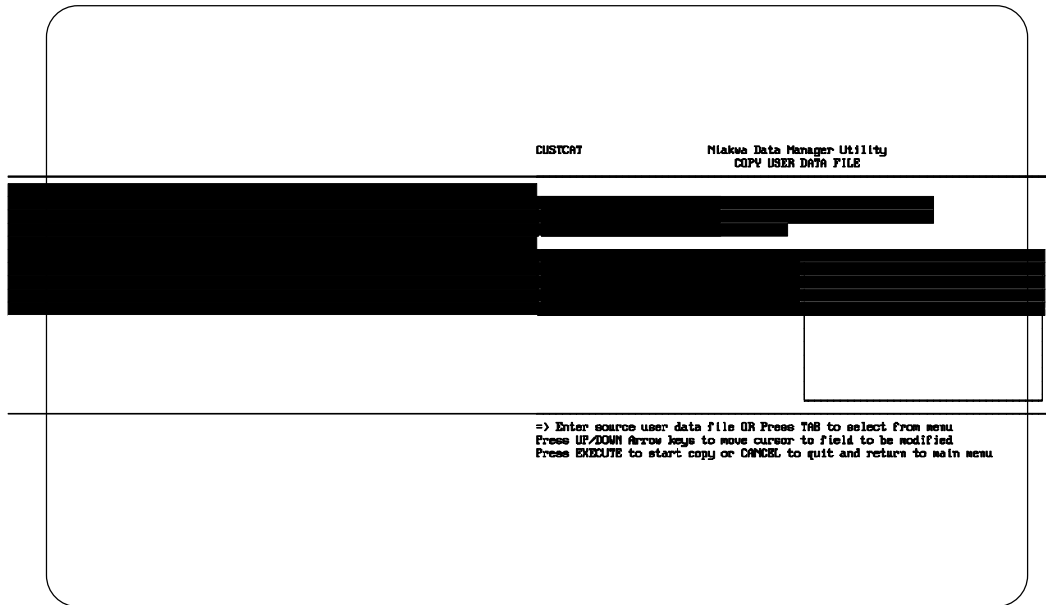


Figure 4 - 25

The copy process has finished when the percent complete field displays 100. Any changes made to the data description or key description files before creating the destination file will be reflected in the new data file

A work file is used for large data files where work space for the conversion is necessary.

NOTE: The source user data file must exist or the operator will be prompted to enter another file.

The destination file must have an entry in the catalog file. If the file has not been created, it will be created by the Utility. If the file already exists, the operator will be asked if the file should be deleted and recreated.

Data may be lost if the destination file does not contain all fields in the source file or if field lengths in the destination file are smaller than in the source file. New fields or additional bytes added to existing fields are initialized to spaces for alpha-numeric fields or zeros for numeric fields.

Press Cancel to return to the Main Menu after the copy process has completed.

4.12 Export Data Dictionary to IQ Utility

The Export Data Dictionary to IQ Utility allows the developer to automatically convert existing NDM data dictionary files to IQ data dictionary files. By using this utility, the developer can immediately use a NDM data file with the IQ program.

NOTE: The IQ development package must be installed on the system where this utility is executed.

The directory where IQ is installed must be on the current path.

There must be sufficient memory available to execute a Basic-2C \$SHELL.

4.12.1 Features

This utility:

- Lists all the data files associated with the current data dictionary.
- Allows the selection of a data file whose data dictionary needs to be converted.
- Converts the NDM data dictionary of the selected file to the IQ data dictionary format.
- Generates the appropriate index for the new IQ data dictionary.

4.12.2 General Use

Select this utility from the Main Menu. Once this utility is selected, a window appears with the names of the data files associated with the current catalog file as shown in Figure 4-26.

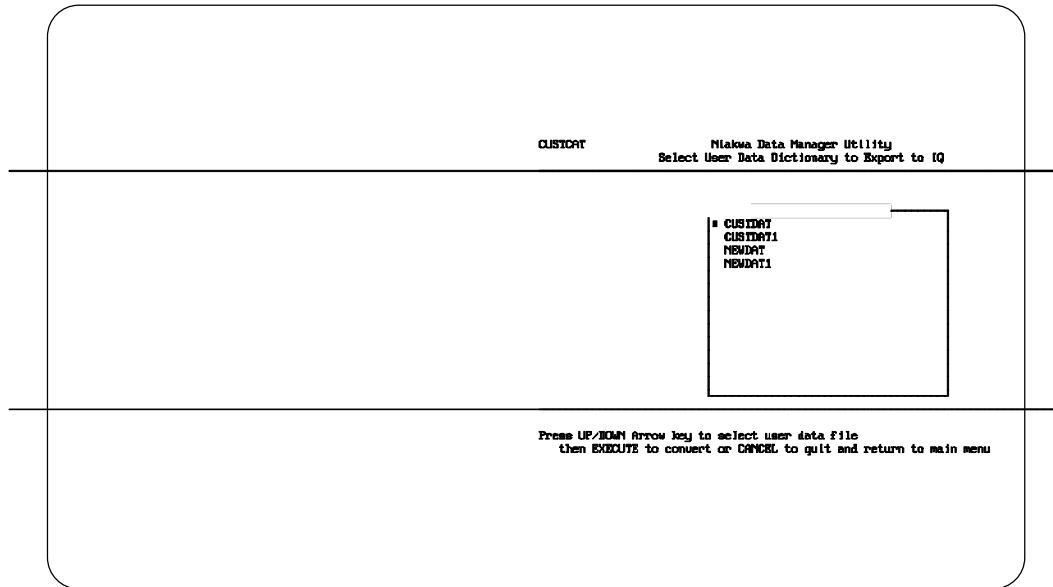


Figure 4 - 26

Use the arrow keys to select a file and press Execute to start the conversion or Cancel to quit.

After selecting a data file whose data dictionary is to be converted to the IQ format, a window appears as in Figure 4-27, prompting the operator for the name of the IQ data dictionary. The default name is DDMASTER.DAT. To select the default press Execute, otherwise enter the full path name of the IQ data dictionary name to be updated or created and press Execute. To quit, blank out the name and press Enter or Execute.

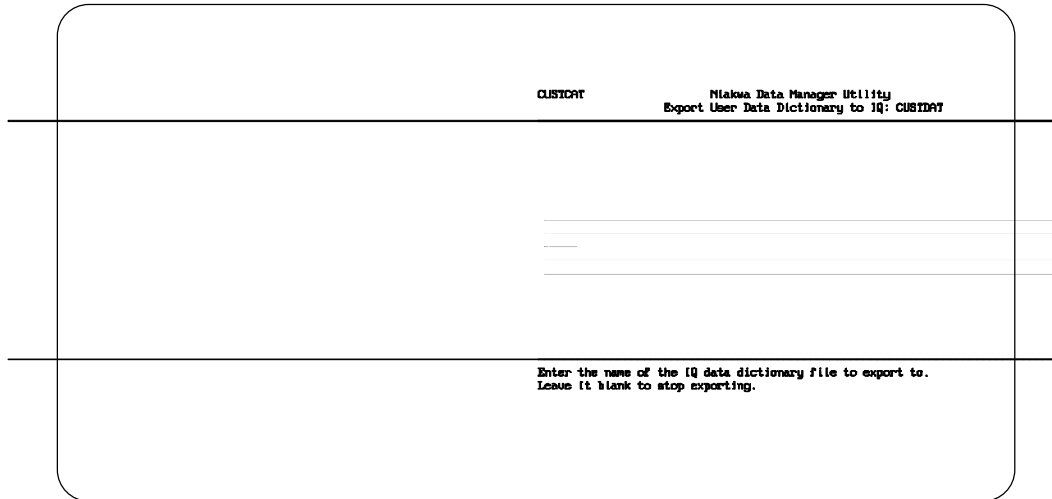


Figure 4 - 27

If the NDM data dictionary file is already defined in the IQ data dictionary file, a window appears as shown in Figure 4-28, asking if the operator wants to replace the existing IQ definition. The default answer is "Y". Press Execute to accept the default or enter "N" and press Execute to not replace the existing information.

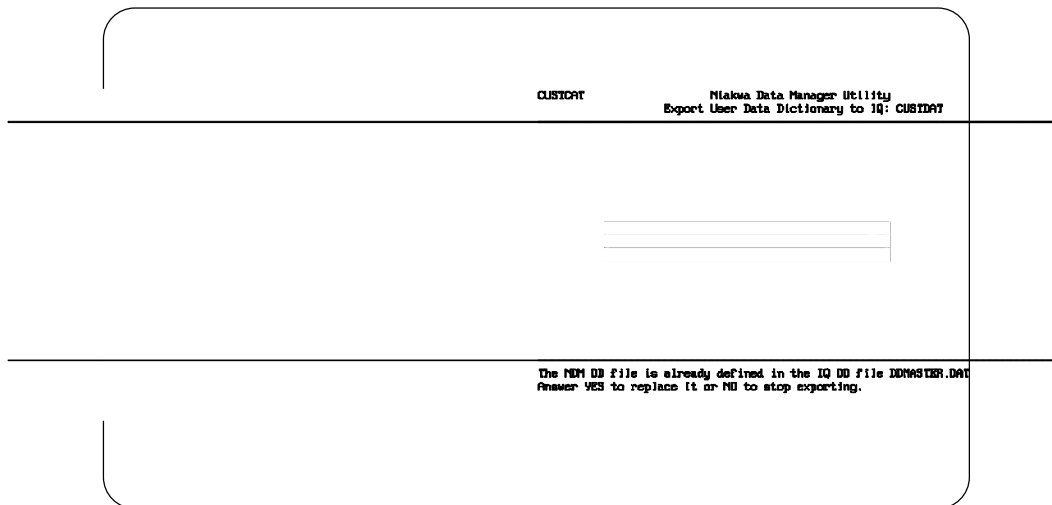


Figure 4 - 28

If the IQ data dictionary file exists, but the file selected is not defined in it, a window appears with a list of database categories. The operator may choose a database category from this list or specify a new database category. If no database categories are defined, the utility displays a prompt for the operator to enter a category. The screen shown in Figure 4-29 appears.

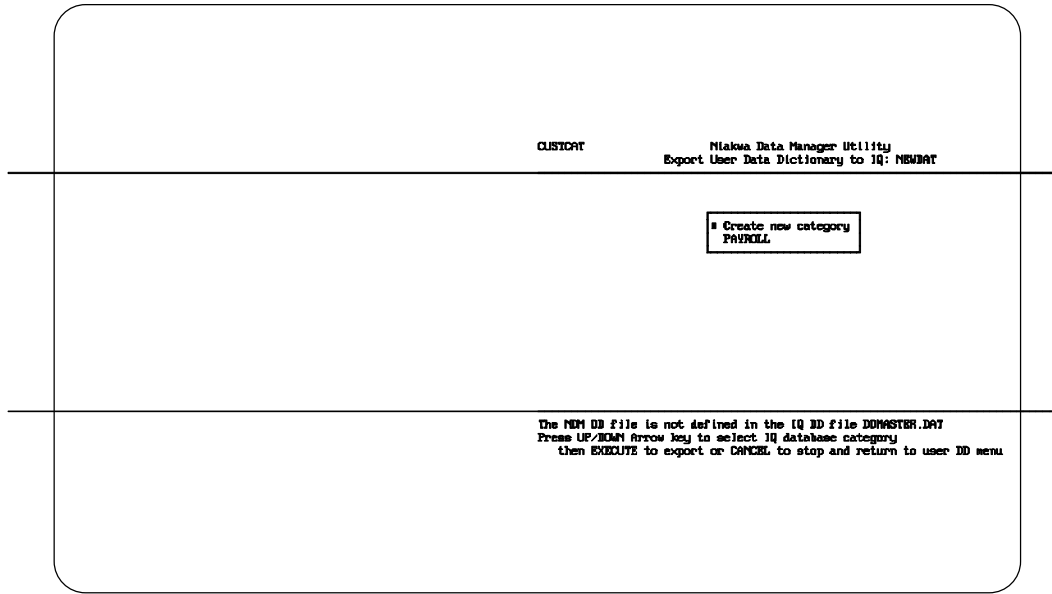


Figure 4 - 29

Enter a database category name in the field provided and press Execute. This database category name is used by the IQ program. Please refer to your IQ manuals for more information.

The exporting process begins and upon completion, the associated IQ data dictionary and index file have been generated automatically. The default names for these files are DDMASTER.DAT and DDMASTER.IDX. The select data file window then reappears. Select another data file/data dictionary to convert or press Cancel to return to the Utilities Main Menu.

4.13 Convert Native File to Basic-2C Format

The Convert Native File to Basic-2C Format Utility allows the operator to convert data in native ISAM files to Basic-2C diskimage format. This is used when it is necessary to convert native ISAM data files from one native ISAM to another.

There is no easy method to convert a data file directly from one native ISAM to another. However, by using a diskimage as an intermediate step, the data can still be transferred and converted. Once transferred, the diskimage file can be converted back by the Convert Basic-2C Format Files to Native File Utility to complete the conversion process and create a data file usable by another native ISAM.

4.13.1 Features

This utility:

- Prompts the user for native File Title and Basic-2C diskimage name.
- Converts the selected files to a Basic-2C diskimage.

4.13.2 General Use

Select this utility from the Utilities Main Menu. A window appears prompting the user for a Basic-2C diskimage name and the native ISAM File Title to be converted.

Enter the name of the diskimage to be used. If the diskimage does not exist it will be created. After entering the diskimage name press Enter.

A window appears with a list of all existing data files in the current catalog file. The user has the choice of entering the name of the data file in the space provided or pressing Tab to select a data file from the file choice window. If the file choice window is used, select the file to convert with the Arrow keys and press Execute. To select all files, select the All Files option.

Once a data file has been selected, press Execute to start the conversion process. The user will be prompted to enter the maximum number of files for the diskimage. Enter the appropriate number and press Enter. A window appears and displays the status of the conversion.

The conversion process is complete when the Percent Complete field displays 100. Press Cancel to return to the Main Menu after the copy process has completed.

4.14 Convert Basic-2C Format Files to Native File

The Convert Basic-2C Format Files to Native File Utility allows the operator to convert data from a Basic-2C diskimage format to native ISAM files. This is used when it is necessary to convert native ISAM data files from one native ISAM to another. This utility is divided into two stages that can be executed at the user's request.

4.14.1 Features

This utility:

- Prompts the user for Basic-2C diskimage name and native File Title.
- Converts the selected files to the selected native ISAM format.
- Allows the User to select either stage of the conversion process.

4.14.2 General Use

Select this utility from the Utilities Main Menu. A window appears prompting the user for a Basic-2C diskimage name, the NDM File Title to be converted, and the stage to execute (1 or 2).

Enter the name of the diskimage to be used. If the diskimage does not exist, the Utility prompts the operator for a valid diskimage. After entering the diskimage name press Enter.

A window appears with a list of all existing data files in the current catalog file. The user has the choice of entering the name of the data file in the space provided or pressing Tab to select a data file from the file choice window. If the file choice window is used, select the file to use with the Arrow keys and press Execute. To select all files, select the All Files option.

Next the operator must select which stage to perform. The two possible stages are described below. Enter a 1 or 2 depending on which stage needs to be executed. Once this selection has been made, press Execute to start the conversion process.

This utility performs in two stages in order to give the user the opportunity to change the data description and key description files before converting the data files. This might be necessary if the field types are not appropriate, or if a key type is not supported on the native ISAM that the files are converted to. Stage 1 is described in section 4.14.3 and stage 2 is described in section 4.14.4.

4.14.3 Stage 1 of Conversion Process

In the first stage the catalog records for the data description file, key description file and the user data file are created.

If the catalog records already exist, the user will be asked if the records should be replaced. If replacement is chosen, then the record is deleted.

Once the record has been deleted, the operator is prompted for the necessary information to be placed in the record for the catalog file. The following information needs to be provided: file name, native ISAM code and other possible native ISAM specific information. See the appropriate Platform Specific Addendum.

After this information has been provided, press Execute and the information will be inserted into the catalog file.

After the catalog information has been placed in the catalog file, the data description file and key description file are created and the corresponding records are inserted into the file.

When stage 1 is completed, a message appears to the user. At this point the user has the choice of continuing with stage 2 or returning to the Main Menu. To return to the Main Menu press Cancel.

4.14.4 Stage 2 of Conversion Process

In stage 2 the user data file is created and the appropriate records are inserted in the file. Once this stage is complete the data file can be accessed by any NDM function calls.

The operator is prompted for the same information as in stage 1. The only difference is that stage 2 must be selected.

When this stage completes executing a message appears informing the user whether the process was complete successfully. To return to the Main Menu press Cancel.

NOTE: The native ISAM version of the user data file is created in the format described by the data description and key description files created by phase 1. If modifications to the data description or key description file are required (due to differences in native field type support between different native ISAMs), these changes must be made prior to executing phase 2.

4.15 Set NDMUTIL Options

The Set NDMUTIL Options Utility allows the operator to select a different catalog file to be used by the utilities. This utility also permits the number of lines per page, the print device, print device address, and the translation name used by the utilities to be modified. The operator can also set the maximum record length of the files being accessed. A default length of 608 bytes has been established.

4.15.1 Features

This utility:

- Displays the current catalog file name, number of lines per page, the current print device number, maximum record length and translation name.
- Allows the above listed options to be modified.

4.15.2 General Use

Select this utility from the Utilities Main Menu. A screen appears with the current values for the catalog file name, number of lines per page, print device number and maximum record length. Refer to the example screen shown in Figure 4-30.

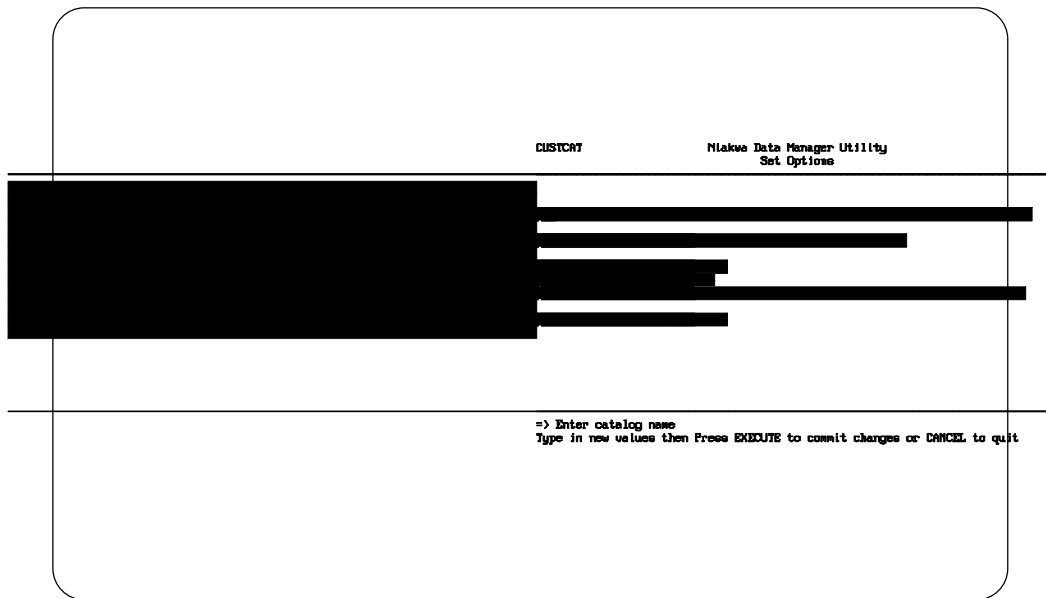


Figure 4 - 30

The catalog field name selected displays in the upper left hand corner of the screen. This name is saved by the utilities and remains the "default" name until it is changed by use of the utility again.

The user may specify the translation table they wish to have in effect. The user may choose from a menu listing all the translations' names in the translation file plus NO TRANSLATION. When the user commits these changes (by using the EXECUTE key to exit) that translation table is put into effect using the **31425 NDM_SET_TRANSLATION_TABLE** call. Also the translation name is stored away and the next time NDMUTIL is used that selected translation table will be in effect.

Press Cancel to return to the Utilities Main Menu.

4.16 Install New Catalog File

The Install New Catalog File Utility creates a new catalog file and copies all existing entries from a specified source Catalog File to the newly created catalog.

NOTE: This must be used prior to creating new data description or key description files for a new application.

4.16.1 Features

This utility:

- Asks for a new catalog file name, relation file name, and source file name.
- Copies all entries from the Source Catalog File.

NOTE: The NDM prevents the operator from modifying the data dictionary system catalog file (NDMCAT). This utility must be used before attempting to create any data description or key description files.

4.16.2 General Use

Select this utility from the Utilities Main Menu. A window appears in which the operator can enter the name of the new catalog file, relation file and source file name as in the example shown in Figure 4-31.

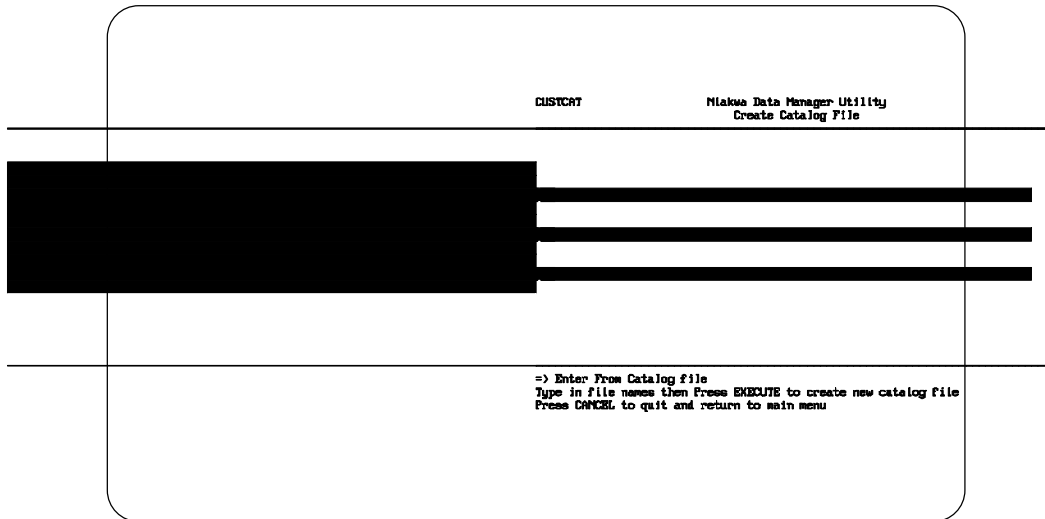


Figure 4 - 31

The Enter or Arrow keys will move between the fields in the window. Press Execute to store the new file names.

This utility copies all entries from the Source to the new catalog file. The operator may then use the other NDM Utilities to create new data dictionaries and key description files for this catalog file.

NOTE: If no user catalog files exist, use the catalog file name NDMCAT as the From Catalog File Name. Use of this file will copy the necessary NDM data dictionary file entries into the new catalog file. NDMCAT is used as the default From Catalog name if this field is left blank.

The new catalog file name displays in the upper left hand corner of the screen. The utilities saves the name of this catalog file and will display it every time the utilities are executed. To change the current catalog file name, use the Set NDMUTIL Options.

To return to the Utilities Main Menu, press Cancel.

4.17 Display Program Information

The Display Program Information Utility displays information about the version of the native ISAM, NDM and NDMUTIL program currently in use.

4.17.1 Features

This utility displays the following information:

- NDMUTIL Version
- Native ISAM in use
- API Version
- API Serial Number
- Number of Active Users
- User Limit
- NDM Directory

NOTE: The above information will vary by operating system and native ISAM.

4.17.2 General Use

Select this utility from the Utilities Main Menu. This utility then displays a screen similar to the example one shown in Figure 4-32.

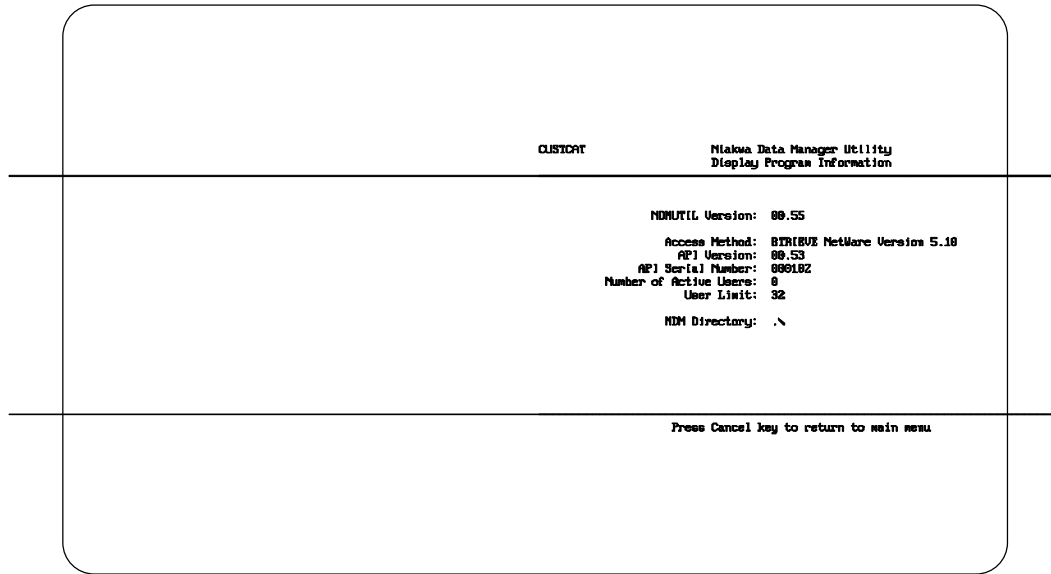


Figure 4 - 32

Press Cancel to return to main menu.

4.18 Translation File Maintenance

This Utility allow the operator to use the translation file and functions.

4.18.1 Features

This utility lets the user add, modify, or delete records in the translation file (NDMTRAN).

4.18.2 General Use

The View Translation File screen lists the translation name field of all the records in the translation file as shown in Figure 4-33.

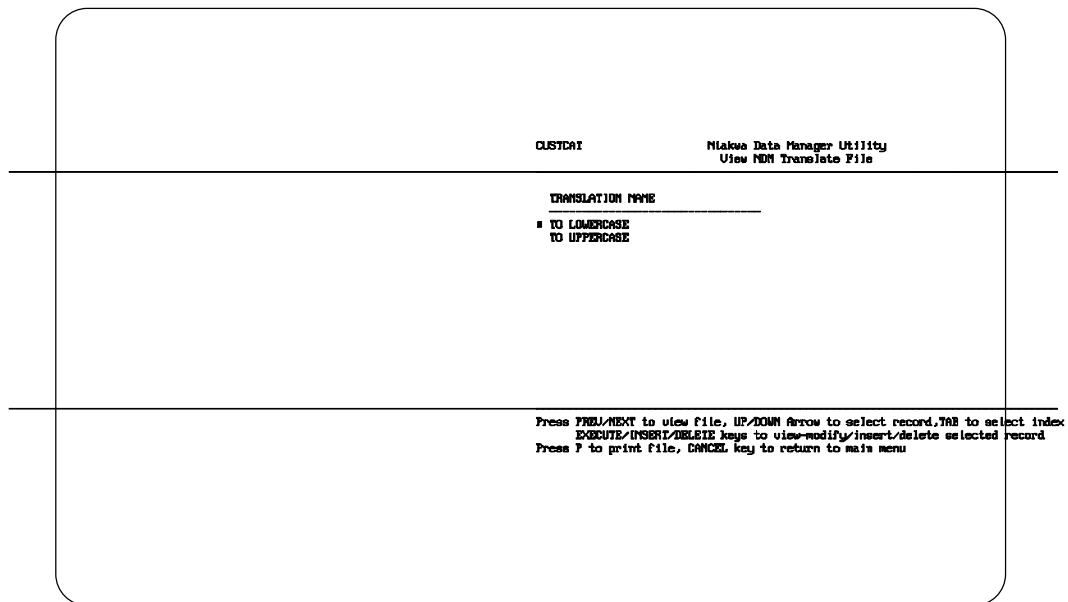


Figure 4 - 33

This screen behaves like the view screen for any of the other files. The insert/update record screen is a two page screen. The first page, shown in Figure 4-34, lets the user enter/modify the Basic-2C to native table entries.

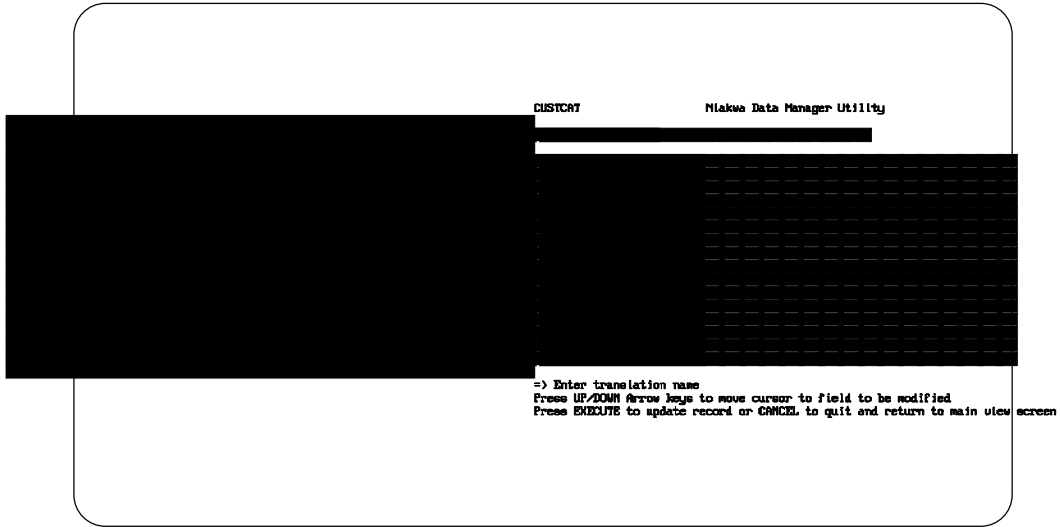


Figure 4 - 34

The second screen, shown in Figure 4-35, lets the user enter/modify the native to Basic-2C table entries.

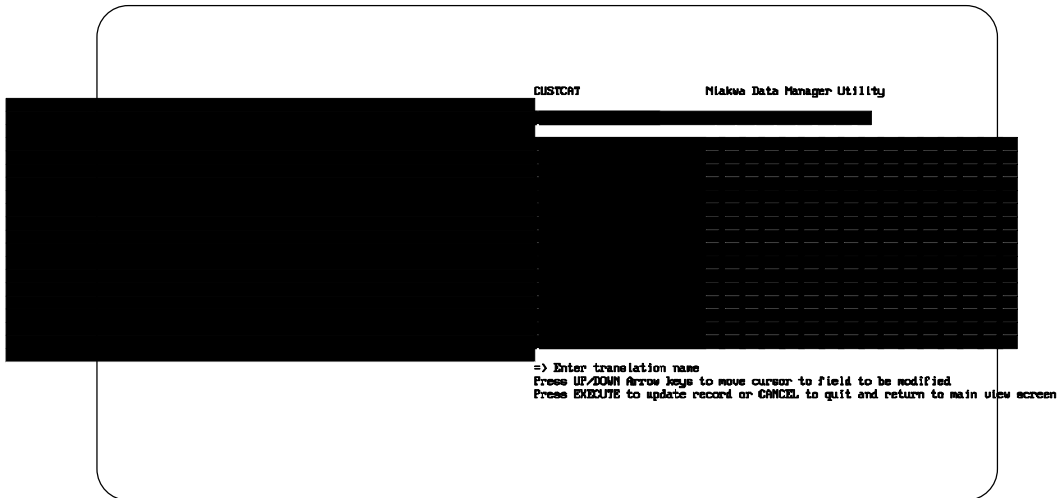


Figure 4 - 35

Both pages let the user enter/modify the translation name. The translation tables consist of 256 1-byte entries. Each entry is 2 hexadecimal digits. The tables are displayed in row-major order, i.e.,

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12...
.
.
.
```

When inserting a new record the initial values for the table entries are setup for no translation.

The translate file entries created by this utility can be used with function call **31030 NDM_CONVERT** to provide automatic translation of Basic-2C field type 21, alpha with translation.

In order to use the translation table created or modified by this utility, the application must use function call **31425 NDM_SET_TRANSLATION_TABLE** and specify the NAME of the entry.

CHAPTER 5

NDM ADVANCED CONCEPTS

5.1 Overview

This chapter details the advanced concepts behind the NDM product.

Section 5.2 discusses NDM multi-user considerations such as file and record locking.

Section 5.3 discusses managing file handles.

Section 5.4 discusses application portability.

Section 5.5 discusses field type conversions.

Section 5.6 discusses the NDM toolbox feature.

Section 5.7 discusses transaction tracking.

5.2 Multi-User Considerations

The NDM fully supports multi-user systems with both file locking and record locking. When a file is locked by a program, no other users can access the records within that file until it becomes unlocked. Record locking permits access to other records stored in the file. The only records that cannot be accessed by other users are those that are locked. The advantage of record locking is that it permits several users to be updating the same file simultaneously.

5.2.1 File Locking

Files may be opened in shared, read only or exclusive mode. Exclusive mode permits exclusive access to the file. When exclusive access is specified, no other user may open the file in any mode. Typically, exclusive mode should only be used for special operations such as initial file creation or possibly for batch update procedures. It also must be used when calling **31080 NDM_CREATE_INDEX** or **31110 NDM_DELETE_INDEX**.

File locking can be specified by the MODE parameter of:

- **31340 NDM_OPEN_FILE**
- **31060 NDM_CREATE_FILE**
- **31070 NDM_CREATE_FILE_FROM_CATALOG.**

File locks are no-wait locks. That is, if an attempt is made to open a file in shared, read only or exclusive mode that is locked by another user, or if an attempt is made to exclusively open a file that is opened by any other user, **31340 NDM_OPEN_FILE** returns an error code of 43 and the file is not opened. The application may then decide how best to proceed (try again, display message to operator, etc.).

5.2.2 Record Locking

Individual records within files may be locked by using the LOCK parameter of:

- **31330 NDM_INSERT_RECORD**
- **31360 NDM_READ_BY_KEY**

- **31370 NDM_READ_BY_POSITION.**

Only one record per open file handle may be locked by each user. An application may successfully lock more than one record per file by use of multiple file handles to access the same file. That is, the same physical file may be opened more than once using different file handles. One record may be locked for each file handle.

A locked record can be read by other tasks using a non-locking read. However, any attempt by other tasks to read with a lock or to rewrite or delete the record will result in an error code 42 being generated by the NDM.

NOTE: Some native ISAMs support passive concurrency. Passive concurrency means the native ISAM detects that another task has modified a record that is being rewritten. With passive concurrency, it is not necessary to lock a record that is to be rewritten. However, since this feature is not supported on all native ISAMs, developers are strongly encouraged to lock any record that is to be rewritten. Refer to Chapter 11, error code 38, for further information on passive concurrency.

Typically, a record should be locked when the application intends to update that record (via **31380 NDM_REWRITE_RECORD**). Locked records are unlocked by:

- **31380 NDM_REWRITE_RECORD**
- **31020 NDM_CLOSE_FILE**
- **31320 NDM_INITIALIZE**
- **31460 NDM_UNLOCK_ALL_RECORD**

When an application is in a loop trying to access a locked record, use of \$BREAK is strongly encouraged to avoid performance degradation.

NOTE: The exact mechanisms used for record locking are controlled by the native ISAM. The developer does not need to be concerned with how record locking is accomplished.



The NDM and the native ISAMs supported by NDM provide no mechanism for detection of deadlock conditions. This is the responsibility of the application. A deadlock condition can occur when two or more terminals attempt to lock the same group of records but in a different order.

5.3 Managing File Handles

The NDM assigns a unique file handle to each file that is opened. The application may specify via **31320 NDM_INITIALIZE** whether the NDM or the application is responsible for storing the entire file handle. If the application specifies that the NDM is to store the file handle, the application must still store a two byte file handle pointer for each file in use and pass this back to the NDM as the `FILE_HANDLE$` parameter wherever required.

NOTE: Whether the application is storing the entire file handle or just the two byte file handle pointers, Niakwa strongly recommends that the application use a single array to store these values and index this array by a logical file number maintained by the application. This idea is discussed further in Chapter 6 of this Programmer's Guide.

5.4 Maintaining Application Portability

The NDM provides several different methods for maintaining application portability. These methods are discussed in the following sections.

5.4.1 Concept of Portability

Applications that conform to NDM standards and recommendations can easily be ported to different hardware or operating system platforms using different native ISAM products. As used in this document, portability means that the application requires little or no modification to the program source code. The NDM does provide features to help in physically porting data files. These features are described in Chapter 4 of this Programmer's Guide.

The NDM provides mechanisms to maintain application portability in three areas:

- 1) File naming and location.
- 2) Use of native field types.
- 3) Use of a common subset of native ISAM functions and features.

5.4.2 File Naming and Location

File naming conventions differ widely among the different native ISAMs and different hardware/operating system platforms supported by the NDM. To allow applications to avoid portability concerns regarding file naming conventions, the NDM supports the use of a logical file title. The logical file title is a 32 character, case sensitive string that allows use of any displayable characters (HEX(10)-(7F)). The file title can be equated to physical file locations by use of the catalog file.

NOTE: The physical file location entered into the catalog file should contain the full path name for the specified file title on the system in use.

Entries in the catalog file can be created and maintained by the NDM Utilities or by the application program itself. A catalog entry for a specific file title can be extracted at execution time by use of the **31150 NDM_GET_CATALOG_ENTRY** function, or the catalog file may be accessed like any other NDM file by the application directly.

5.4.3 Native Field Types

Different native ISAMs have differing limitations on the types of fields that may be used as index segments. Using a field type as an index segment that is not supported on a particular native ISAM can prevent the application from operating properly on that native ISAM. The NDM provides several features designed to preserve application portability relating to this issue.

By default, the NDM supports only four field types as valid key segments:

- Alphanumeric (may be used for any binary fields).
- Integer (for numeric values).
- Date
- Time

Application developers are advised to limit their use of index segment field types to these. However, the NDM does provide a method of using other field types as keys where supported by a particular native ISAM. The **31420 NDM_SET_TOOLBOX_STATUS** function allows the developer to enable other field types as index segments if needed by the application.

The toolbox feature should only be used when necessary to access a file that is main-



tained by a non-NDM application.

If **31420 NDM_SET_TOOLBOX_STATUS** has not been called to enable the use of extended key types, the NDM returns error code 35 when attempting to call **31400 NDM_SET_CURRENT_INDEX** to set an index containing a key type other than alphanumeric or integer. This error can also occur on functions that set the current index implicitly. These include:

- 31060 NDM_CREATE_FILE
- 31070 NDM_CREATE_FILE_FROM_CATALOG
- 31340 NDM_OPEN_FILE

Field Type Conversion

The NDM provides a mechanism by which different Basic-2C field types can be converted to supported native field types at execution time. This allows the Basic-2C application to process and manipulate data in a consistent manner despite the actual format used to store the data. Refer to Section 5.5 for further details on the field type conversion facilities.

5.4.4 Common Native ISAM Features

Different native ISAM products have different features and limitations. To ensure application portability, Niakwa has implemented a subset of native ISAM features in the NDM. This subset of features should be more than adequate for most applications, but to accommodate special requirements, the NDM supports a group of extended features for each native ISAM supported.

These features must be explicitly enabled by the application by using the **31420 NDM_SET_TOOLBOX_STATUS** function. Attempts to use any extended feature not enabled by **31420 NDM_SET_TOOLBOX_STATUS** results in a non-zero return code

(error). Refer to the NDM Platform Specific Addendum for a list of extended features supported for the specific native ISAM.

NOTE: Niakwa strongly encourages developers to avoid using extended features for development of application packages intended for use on multiple platforms.

5.5 Field Type Conversions

Basic-2C applications may use a variety of field types. Many of these field types are highly specific to Basic-2C and are not directly supported by native ISAMs.

Although it is possible to store the Basic-2C data types unmodified under the native ISAM (as alphanumeric), this is not recommended for two important reasons:

1. Fields used as key segments must conform to a supported native field type format. It is not possible, for example, to define a Basic-2C packed decimal field as a native alphanumeric field and expect it to work properly as a key segment. Records would be stored in an incorrect logical order because the collating sequence required for packed decimal fields differs from that required for alphanumeric fields -- alphanumeric fields are collated one byte at a time from left to right; packed decimal fields must be collated on the implicit value contained in the field, not the binary value of the actual bytes.
2. Data interchange with third party products and applications. A primary feature of the NDM is the data interchange capabilities. With proper use of the NDM, data stored by Basic-2C applications can be directly accessed by third party products that use the same native ISAM as the NDM on the platform in use. Correspondingly, Basic-2C applications can directly access any data stored either by other Basic-2C applications using the NDM or by third party applications.

However, for data interchange to work properly, the different applications not only have to be able to access the data at the record level, they need to be able to understand the contents of individual fields.

To provide a portable method of using native field types, the NDM provides a mechanism by which Basic-2C field types can be converted to and from native field types dynamically at execution time.

The remainder of this section discusses several topics related to this Field Type conversion. Refer to Section 3.3 of this Programmer's Guide for a complete description of Basic-2C and native field types supported by NDM.

5.5.1 The Conversion Table

The conversion table is an internal table stored by NDM that contains the specification for conversion of all fields in a data buffer from Basic-2C to native ISAM field types and native ISAM back to Basic-2C field types. The conversion table contains information about each field including:

- Basic-2C Field Type
- Basic-2C Start Position
- Basic-2C Field Length
- Basic-2C Decimal Position
- Native Field Type
- Native Start Position
- Native Field Length
- Native Decimal Position

A specified conversion table is used by **31030 NDM_CONVERT** to convert the specified data buffer between Basic-2C and the native format (as specified by the direction parameter). The conversion table is designed to allow relocation of fields within the data buffer as well as conversion of the data within the fields.

5.5.2 Establishing the Conversion Table

Conversion tables cannot be directly created or modified by the application program. Rather, the API functions **31050 NDM_CREATE_CONVERSION_TABLE** or **31040 NDM_CREATE_CONV_TABLE_FOR_KEY** can be used to create conversion tables. Both API functions require that a NDM data description file containing definitions of the Basic-2C and native field types, start, length, and decimals for each field be present and open.

The NDM data description file allows specification of a conversion table number for each field. The conversion table number is used to establish different conversion tables for the same file. Typically, this capability is used only when the application needs to convert a subset of fields for a particular record or when some overlapping fields are defined.

NOTE: For most applications, specifying conversion table number zero for all fields will be adequate.

The **31050 NDM_CREATE_CONVERSION_TABLE** function is used to create the actual internal conversion table used by **31030 NDM_CONVERT**. A conversion table number may be specified when calling **31050 NDM_CREATE_CONVERSION_TABLE**. **31050 NDM_CREATE_CONVERSION_TABLE** then creates the internal conversion table for all fields defined with the specified conversion table number.

For example, assume that a NDM data description contains the information in Table 5-1.

Field Name	Conversion Table Number	B2C Field Type	B2C Start Position	B2C Length	Native Field Type	Native Start Position	Native Length
CUSTOMER NUMBER	0	7	1	5	1	1	5
NAME	0	7	6	20	1	6	20
NAME-I	1	7	6	5	1	6	5
PHONE	0	7	26	13	1	26	13
AREA-CODE	2	7	27	3	1	27	3
YTD-SALE\$\$	0	5	39	5	8	39	5
NUMBER OF SALES	0	6	44	3	2	44	2
CITY	0	7	47	20	1	46	20

Table 5-1

Field Types used in Table 5-1 are:

Basic-2C:

7 = Alphanumeric

5 = Packed Decimal (equivalent to \$PACK 5dxx)

6 = Unsigned Packed Decimal (\$PACK 6dxx)

Native:

1 = Alphanumeric

8 = Packed Decimal (could be identical with Basic-2C Packed Decimal)

2 = Integer

In this example, the fields NAME-I and AREA-CODE are fields that overlap other fields (NAME and PHONE respectively) and are defined strictly for use as index segments. Therefore, they should NOT be converted as part of the standard record. The assignment of conversion table numbers 1 and 2 will accomplish this.

A call to **31050 NDM_CREATE_CONVERSION_TABLE** specifying conversion table number zero and specifying this data description file creates an internal conversion table for the basic record for this file containing a conversion specification for:

```
CUSTOMER NUMBER
NAME
PHONE
YTD SALES$
NUMBER OF SALES
CITY
```

For the convenience of the application developer, a special function has been included to allow easier conversion of key fields. A key field must be passed to **31360 NDM_READ_BY_KEY**. This key field must be converted to native format and placed in a record buffer in its proper position before calling **31360 NDM_READ_BY_KEY**.

31040 NDM_CREATE_CONV_TABLE_FOR_KEY creates a special internal conversion table for a specified key allowing the application to create a key field starting at

byte one of a string variable. Alternatively, by specifying **KEY_IN_RECORDS** as "Y", the application must still pack the key field into the record buffer at its proper location before calling **31040 NDM_CREATE_CONV_TABLE_FOR_KEY**. Here, the advantage of using **31040 NDM_CREATE_CONV_TABLE_FOR_KEY** as opposed to **31050 NDM_CREATE_CONVERSION_TABLE** is strictly for performance because it converts fewer fields. **31040 NDM_CREATE_CONV_TABLE_FOR_KEY** requires the presence of a key description file. It will extract all FIELDNAMEs defined as segments for the specified index number from the key description file and then access the data description file to extract field type, start, length, and decimal information for each key field.

For example, assume that for the above data description, a key description file is defined as in Table 5-2.

Index Number	Segment Number	Fieldname	Unique	Ascending
1	0	CUST #	Y	
1	1	CUSTOMER NUMBER		Y
2	0	SHORT NAME	N	
2	1	NAME-I		Y
3	0	SALES BY AREA	N	
3	1	AREA-CODE		Y
3	2	YTD SALESS		N

Table 5-2

Conversion tables for each individual index can be created by repeated calls to **31040 NDM_CREATE_CONV_TABLE_FOR_KEY**.

5.5.3 When To Call 31030 NDM_CONVERT

Whenever field level data is passed between Basic-2C and the NDM, it should be converted by **31030 NDM_CONVERT**. Table 5-3 shows all API functions that require conversion and where and when **31030 NDM_CONVERT** should be used.

NOTE: The direction stands for: 1 is Basic-2C to native; -1 is native to Basic-2C:

API Function	Before/After	Direction
31010 NDM_APPEND_UNIQUE_KEY_RECORD	Before	1
31330 NDM_INSERT_RECORD	Before	1
31360 NDM_READ_BY_KEY	Before	1 (for the key)
31360 NDM_READ_BY_KEY	After	-1 (for the record)
31370 NDM_READ_BY_POSITION	After	-1
31380 NDM_REWRITE_RECORD	Before	1

Table 5-3

For all cases except the convert before **31360 NDM_READ_BY_KEY**, a complete record conversion should be used. For the convert before **31360 NDM_READ_BY_KEY**, a conversion table specific to the key field being passed may be used (as created by **31040 NDM_CREATE_CONV_TABLE_FOR_KEY**).

5.5.4 Performance Issues

The NDM field type conversion routines are heavily optimized to minimize the impact on overall application performance. In cases where no actual conversion is required, the NDM detects this condition and simply skips the field.

Below are some guidelines that will help minimize the performance overhead associated with field type conversion:

1. Wherever possible, use identical or similar Basic-2C and native field types. For example, for Basic-2C packed decimal fields, assign the native type as packed decimal (unless this field must be a key). On some native ISAMs (Btrieve and SDtrieve), the native packed decimal field type is identical with the Basic-2C packed decimal, thus requiring no actual conversion. For another example, use Basic-2C signed binary (\$PACK C0xx) as an equivalent for native integer wherever possible (conversion here is usually just a matter of byte swapping).

2. Do not arbitrarily change the order or size of fields between the Basic-2C record and the native record. Relocation or resizing of fields adds to the time required by the conversion routines. In particular, avoid changing the size of numeric fields even for like field types. Resizing of numeric fields can be expensive in terms of performance.
3. Avoid the use of any floating point format. Conversion to and from floating point formats is the most expensive in terms of time required. If using floating point formats cannot be avoided, attempt to use an identical native floating point format. Btrieve, for example, supports both single and double precision floating point formats directly equivalent to the Basic-2C \$PACK F304 and F308 formats.

5.5.5 Managing Conversion Table Handles

Both **31040 NDM_CREATE_CONV_TABLE_FOR_KEY** and **31050 NDM_CREATE_CONVERSION_TABLE** return a two byte conversion table handle to the application where this handle is actually a pointer to the address in memory where the necessary information is stored. This conversion table handle is unique for each conversion table created. The conversion table handle for the conversion table to be used must be specified by the application when calling **31030 NDM_CONVERT**.

The best way for applications to manage these conversion table handles is to store them in a single array (C\$() for example). If **31040 NDM_CREATE_CONV_TABLE_FOR_KEY** is not being used, then this array can be a single dimensional array with the number of elements being equal to the maximum number of files the application will ever have open concurrently. If **31040 NDM_CREATE_CONV_TABLE_FOR_KEY** is being used to create separate conversion tables for each index for each file, then the conversion table array should be dimensioned as a two dimensional array:

```
DIM C$(X,Y)2
```

where X is the maximum number of files that the application will have open concurrently and Y is the maximum number of indices per file plus one for the record itself. The maximum number of indices per file supported by the NDM is 9 unless the toolbox extensions are used.

NOTE: It is recommended that all required conversion tables for a file be created at the time the file is opened and destroyed (via 31130 NDM_DESTROY_CONVERSION_TABLE) at the time the file is closed. The file number and index number, if 31040

NDM_CREATE_CONV_TABLE_FOR_KEY is used, should be used as indices to the conversion table array. This concept is discussed further in Chapter 6 of this Programmer's Guide.

5.6 Toolbox Features

Most native ISAMs provide features that are not supported by all NDM supported native ISAMs. Therefore, to maximize portability of NDM applications, these features are not normally supported under NDM, but applications unconcerned about portability may make use of these features through a NDM feature called the "toolbox".

The toolbox consists of a set of NDM features, each of which may be individually enabled or disabled through a call to a special API function **31420 NDM_SET_TOOLBOX_STATUS**. Therefore, non-portable applications are easily identifiable by their call(s) to this special API function. The availability of each toolbox feature can be determined by calling another API function **31310 GET_TOOLBOX_STATUS**.

Below is a list of the toolbox features that the NDM attempts to support for all native ISAM's.

NOTE: Some native ISAMs do not support all these features - trying to use them will result in an error code.

5.6.1 Filename Extensions

When this feature is enabled, the application may specify a filename with an extension to such API functions as **31340 NDM_OPEN_FILE** or **31060 NDM_CREATE_FILE**. Normally, the NDM requires that no extension be present since it supplies its own. However, an application may wish to open valid native ISAM files that were not created under the NDM.

To specify a filename with no extension but to prevent NDM from appending the default extension to that name, the file name should end in a period.

5.6.2 Key Types

The NDM key segments can normally be one of four types: string, two-byte integer, date, or time. When the "key types" toolbox feature is enabled, all key types supported

by the native ISAM may be used. These types are listed in the "field type" file included with the NDM system.

For details on the format of each native ISAM specific field type, refer to the native ISAM documentation.

5.6.3 File Limits

To ensure portability of NDM applications, any limits such as maximum data record length are restricted to the minimum value supported across all native ISAMs. By enabling the "file limits" toolbox feature, an application can extend these limits to the maximum allowable values for the native ISAM in use. There is also an API function, **31260 NDM_GET_LIMIT_HIGHWATERS**, which reports, for each limit, the maximum value used by the application since the last call to the initialization routine. These limits are defined in Table 5-4.

Description	NDM non-extended limits	Btrieve	SDtrieve	C-ISAM
Record length	32511	32767	32767	32511
Key length (bytes)	120	255	167	120
No. of keys/file	9	24	9	no limit
No. of segs/key	8	24	24	8

Table 5-4

NOTE: The maximum number of key segments that may be defined for a file is fixed at 24.

5.6.4 Transaction Processing

This feature allows access to the:

- **31450 NDM_TRANSACTION_START**
- **31440 NDM_TRANSACTION_COMPLETE**
- **31430 NDM_TRANSACTION_ABORT**

calls. These are used to logically group a series of file operations so that they are treated as a single indivisible operation.

For more information on transaction tracking, refer to Section 5.7.

5.6.5 Get/Set Position

Allows access to the **31280 NDM_GET_POSITION** and **31410 NDM_SET_POSITION** calls, which an application can use to "remember" the current record of a file for later restoration.

5.6.6 Create/Delete Index

Allows access to the **31080 NDM_CREATE_INDEX** and **31110 NDM_DELETE_INDEX** calls, which are used to create temporary indices for existing data files.

This feature can be very useful as a replacement for sort routines. Based on the fields to be used for the sort, a temporary index can be created using the fields as segments of the temporary index. Each segment may be defined as ascending or descending depending on the desired sort order.

Once the index has been created, the application can simply select that index, position to the start of the file, and read sequentially through the file to retrieve records in sorted order.

NOTE: The temporary index should be deleted after the sort file is processed.

5.7 Transaction Tracking

The NDM provides a feature for grouping related file access calls, called transaction tracking. The purpose of using such a feature is to preserve data integrity. For example, if a group of NDM calls in an application were updating data files, transaction tracking would be used to group the calls into a single transaction. Then any changes to the data files would not be made permanent until all calls were successfully completed.

NOTE: Transaction tracking is not supported by all native ISAMs. As such, transaction tracking is a toolbox feature. Before using any of the transaction tracking calls, the

transaction toolbox function (feature # 4) must be enabled. Refer to your native ISAM manual to determine if transaction tracking is supported.

5.7.1 Transaction Tracking Statements

Transaction tracking is accomplished through a series of three NDM calls, **31450 NDM_TRANSACTION_START**, **31440 NDM_TRANSACTION_COMPLETE**, and **31430 NDM_TRANSACTION_ABORT**.

The **31450 NDM_TRANSACTION_START** call defines the start of a transaction. To end a transaction the application calls **31440 NDM_TRANSACTION_COMPLETE**, which completes the transaction by making permanent all changes to the data files since the last **31450 NDM_TRANSACTION_START** call. If the call to **31440 NDM_TRANSACTION_COMPLETE** is called without being preceded by a **31450 NDM_TRANSACTION_START** call, an error code is returned.

The **31430 NDM_TRANSACTION_ABORT** call will cancel all NDM operations performed since the last **31450 NDM_TRANSACTION_START** statement. The purpose of this call is to allow the application a method of aborting if an unexpected situation occurs while accessing data files. For example, if an application is using two data files, and has already updated the first file, but could not access the second file, a call to **31430 NDM_TRANSACTION_ABORT** would automatically stop the transaction and thus reverse all changes made to the first data file during the transaction.

NOTE: If the system is shut down abnormally (i.e., power failure, program failure) during a transaction, the next time the system is started up, an implicit 31430 NDM_TRANSACTION_ABORT call is performed. This protects data files from being in a partially updated state.

CHAPTER 6

NDM PROGRAMMING

6.1 Overview

This chapter discusses the basic techniques of programming in Basic-2C using the NDM API functions to manage the application's data.



This cannot be viewed as a complete description of potential programming techniques. The use of the NDM is only limited by the creativity of the programmer.

The NDM allows developers many choices regarding application design. Some of these choices include:

- Use of the NDM field type conversion feature.
- Use of the NDM data dictionary files such as the catalog file, data description file, key description file, and relation file.

- Use of straight line code to call the API functions versus generalized subroutines for common API functions.

NOTE: Examples in this chapter include generalized routines that use the NDM data dictionary files and field type conversions. The examples given in this section are code segments and are not intended to form executable, working programs. These are included as a guide for proper NDM programming techniques. Several working example programs are included in the NDM Development Package. Refer to Chapter 9 for details. It is recommended that the developer review the complete list of API functions in Chapter 10 before reading this chapter.

Section 6.2 discusses the basic NDM statements.

Section 6.3 discusses the NDM startup routines.

Section 6.4 discusses the NDM shutdown routines.

Section 6.5 discusses creating data files using the NDM Utilities.

Section 6.6 discusses opening data files.

Section 6.7 discusses methods for selecting an index.

Section 6.8 discusses reading records by key.

Section 6.9 discusses reading records sequentially from the start of a file and from a key value.

Section 6.10 discusses adding new records to a data file.

Section 6.11 discusses updating records.

Section 6.12 discusses deleting records.

Throughout this chapter, reference is made to an example file titled CUSTOMER. The data description and key description file layouts for this file are shown below. For generalized layout information on these data dictionary files, please refer to Chapter 3 of this Programmer's Guide.

Data description file layout (file title is CUSTOMER DD) is described in Table 6-1:

Field Name	Conversion Table Number	B2C Field Type	B2C Start Position	B2C Length	Native Field Type	Native Start Position	Native Length
CUSTOMER NUMBER	0	7	1	5	1	1	5
NAME	0	7	6	20	1	6	20
NAME-I	1	7	6	5	1	6	5
PHONE	0	7	26	13	1	26	13
AREA-CODE	2	7	27	3	1	27	3
YTD SALE\$\$	0	5	39	5	8	39	5
NUMBER OF SALES	0	6	44	3	2	44	2
CITY	0	7	47	20	1	46	20

Table 6-1

Field types used in Table 6-1 are:

Basic-2C:

- 7 = alphanumeric
- 5 = packed decimal (equivalent to \$PACK 5dxx)
- 6 = unsigned packed decimal (equivalent to \$PACK 6dxx)

Native:

- 1 = alphanumeric
- 8 = packed decimal (may or may not be identical to Basic-2C Packed decimal)
- 2 = integer

The key description file layout (file title is CUSTOMER KD) is described in Table 6-2:

Index Number	Segment Number	Fieldname	Unique	Ascending
1	0	CUST #	Y	
1	1	CUSTOMER NUMBER		Y
2	0	SHORT NAME	N	
2	1	NAME-I		Y
3	0	SALES BY AREA	N	
3	1	AREA-CODE		Y
3	2	YTD SALES\$		N

Table 6-2

6.2 Basic NDM Statements

Table 6-3 shows the basic file access functions and the corresponding API function calls:

Basic Function	API Function Call
Initialize NDM	31320 NDM_INITIALIZE
Create a File	31060 NDM_CREATE_FILE 31070 NDM_CREATE_FILE_FROM_CATALOG
Open a File	31340 NDM_OPEN_FILE
Read a Record	31360 NDM_READ_BY_KEY 31370 NDM_READ_BY_POSITION
Delete a Record	31120 NDM_DELETE_RECORD
Write a New Record	31330 NDM_INSERT_RECORD
Update an Existing Record	31380 NDM_REWRITE_RECORD
Close a File	31020 NDM_CLOSE_FILE

Table 6-3



Examples in this chapter use both the number and name of the API function. The function name is used for documentation purposes only. Function names should not be used in actual code, only the function numbers. In addition, long variable names are shown for the parameters. This is for documentation purposes only. When programming, use standard Basic-2C variable formats.

6.3 Startup Routines

It is recommended that the developer design a single startup routine for each application developed that uses the NDM. This startup routine typically performs the following functions:

- Determine the parameter sizes for the application. This includes:
 - Maximum number of files to be opened concurrently.
 - Maximum number of conversion tables and conversion table entries to be used.
 - Maximum record length to be used.
- Initialize the NDM via **31320 NDM_INITIALIZE**.
- Optionally extract the default NDM directory via **31160 NDM_GET_CONFIGURATION**.
- Optionally open the catalog file to be used by the application.
- Optionally enable any extended features that are to be used through **31420 NDM_SET_TOOLDOT_STATUS**.

NOTE: In the examples in this chapter, extended key types are used. Refer to Section 6.3.5 or Chapter 5 for more information.

- Optionally dimension common variables to be used by the subsequent modules.
- Optionally get file handle size via **31230 NDM_GET_HANDLE_SIZE**.

6.3.1 Determining Parameter Sizes

The methods used to determine the parameter sizes vary widely from one application to the next. For applications with a set number of files and fields, this information can often be hard coded. For data base style applications, the application must have some method of soft coding this information (a user modifiable table that is read by the startup routines is one approach).

For example purposes, assume that our application has hard coded these values. The values used are:

Maximum Number of Open Files = 23

This value is based on the assumption that a maximum of 20 application data files are opened concurrently. Three work files are also required. These work files are: the catalog file and a data description and key description file that are opened temporarily when opening a data file. At various times, the application needs these files opened to get information needed for different API function calls; for example, **31040 NDM_CREATE_CONV_TABLE_FOR_KEY** and **31150 NDM_GET_CATALOG_ENTRY**.

Maximum Number of Conversion Tables = 200

This value is based on the assumption that separate conversion tables are used for each index and that there are no more than 9 indices per file (this is the maximum number of indices supported by the NDM without using the extended features).

Maximum Number of Conversion Table Entries = 500

This value represents the total number of fields in all files that may be open concurrently plus the number of segments in all indices. If the application does not use the field type conversion feature of the NDM, the number of conversion tables and conversion table entries is zero.

Maximum Record Length = 1024

This value must be at least equal to the length of the largest record used by the application.

6.3.2 Initialization of NDM

Before any other API function can be called, the NDM must be initialized via the **31320 NDM_INITIALIZE** call.

The values determined in Section 6.3.1 for number of open files, conversion tables, conversion table entries, and maximum record length should be passed to **31320 NDM_INITIALIZE**.

For the FILE-COUNT parameter, the programmer must choose whether the NDM is to store the file handles or whether the application stores these itself. Typically the application allows the NDM to store these, and therefore it specifies the FILE-COUNT parameter as the maximum number of concurrently open files.

In cases where the application must store the file handles, the application should determine the file handle size via **31230 NDM_GET_HANDLE_SIZE** (after calling **31320 NDM_INITIALIZE**) and dimension the variable(s) it uses to store the file handles based on the returned value.

NOTE: Programmers may want to store the file handles on disk on systems where there are severe memory limitations and the application uses many files.

If file handles are stored by the NDM, the NDM must allocate a fixed amount of memory based on the file handle size times the number of file handles. This memory then remains permanently allocated to the NDM until the next call to **31320 NDM_INITIALIZE**.

The amount of memory used for other **31320 NDM_INITIALIZE** parameters also warrants discussion. This includes:

- Each conversion table requires 10 bytes.
- Each conversion table entry requires 18 bytes.
- The maximum record length requires the size specified.

Although the total memory requirements for these parameters is low for most applications (under 10K), the developer should not use large values for these parameters to avoid unnecessary memory allocation by the NDM.

6.3.3 Get Default NDM Directory

A default directory location for the NDM files may be defined by the NDM environment variable. The exact statements required for setting the NDM environment variable vary from one operating system to the next. Refer to the Platform Specific Addendum for more information.

The default NDM directory location specified by the NDM environment variable may be extracted as the `NDM_DIRECTORY$` parameter of **31160 NDM_GET_CONFIGURATION**. This may then be used by the application to locate any files stored in this directory.

NOTE: The examples in this chapter use the NDM default directory to locate the catalog file. They also assume that the full path name for all other files is defined via the catalog entries. In examples where the catalog file is not used, the examples assume that all files are located in the current directory.

6.3.4 Open Catalog File

The developer may choose to use individual catalog files for each application or to use a single catalog file for all NDM files on a given system. The examples assume that a single NDM catalog file, `SAMPCAT`, is to be used and that this is located in the NDM default directory.

Typically, if a catalog file is to be used, it should be opened in the startup routines and left open for the duration of the application.

6.3.5 Enable Extended Features

If any extended features are to be used, it is strongly recommended that **31420 NDM_SET_TOOLBOX_STATUS** be called in the startup routines. The examples in this chapter use extended key types. For more information on the NDM toolbox feature, refer to Section 5.6 later in this chapter.

6.3.6 Define Variables

Variables to be used with the API functions should be defined by the developer in the startup routines. These variables should include:

- An array for storing file handles.

- An array for storing conversion table handles.
- A record buffer variable.

For applications that integrate the startup routines with every module, these variables may be non-common. For applications that execute the startup routines only once and then overlay a series of individual application routines (the recommended technique), use of common variables is recommended.

6.3.7 Sample Program for Startup Routines

Example 6.1 makes these assumptions:

- The application uses a fixed number of files with a fixed number of fields.
- The NDM stores the file handles.
- Extended key types are used.
- The catalog file is stored in the NDM default directory.
- The startup routines are executed only once and allows overlay of different application modules (as executed from "MENU").

Example 6.1 - Sample Startup Routine

```

0010 REM MODULE START - SAMPLE NDM STARTUP ROUTINE
0020 COM F$(23)2 : REM file handle array
      : REM maximum of 20 open files for the application plus three
      : REM extra for NDM files (catalog file, and a data description
      : REM and key description file opened temporarily for each data
      : REM file)
      : REM defined with length of 2 because NDM will be storing
      : REM file handle and only needs to store the two byte pointer
0030 COM C$(20,10)2 : REM conversion table handle array
      : REM for 20 files with 9 indices per file (element X,1 is for
      : REM the main record)
0040 COM R$1024 : REM record buffer - maximum record length =
      1024
0050 DIM A$(4)72 : REM work variable used for 31160 NDM_GET_CON-
      FIGURATION
0055 DIM A$72 : REM work variable for catalog Filename
0060 A=200 : REM conversion tables
0070 B=500 : REM conversion table entries
0080 C=23
      : REM file count - 20 files for the application plus three
      : REM work files
0090 D=1024 : REM record length
0100 GOSUB'31320 NDM_INITIALIZE(A,B,C,D,R)
0110 IF R<>0 THEN GOSUB 'xxxx(R,"Cannot initialize NDM")
      : REM always check the return code!
      :

```

```

REM The application should have a general routine for handling
fatal errors. This routine('xxxx') is not included in the sample
programs.
0120 GOSUB'31160 NDM_GET_CONFIGURATION
(X,X,X,X,AS(1),X,AS(2),AS(3),U1,U2,AS(4),R)
: IF R<>0 THEN GOSUB'xxxx(R,"Cannot get configuration")
0125 GOSUB'31420 NDM_SET_TOOLBOX_STATUS(2,1,R)
: REM Allow extended key types
: IF R <>0 THEN GOSUB'xxxx(R,"Cannot enable extended key
types ")
0130 AS=AS(3)&"SAMPCAT" : REM build filename for Sample
catalog file
0140 GOSUB'31340 NDM_OPEN_FILE(AS,"S",1,F$(21),R)
: REM open catalog file, in shared mode; set current index to
1; use file number 21
: IF R <>0 THEN GOSUB' xxx(R,"Cannot open SAMPCAT.DAT")
0150 LOAD T"MENU" : REM application main menu program

```

6.4 Shutdown Routines

Whenever the NDM processing for an application is complete, shutdown of the NDM must be done properly. A proper shutdown ensures that all open files are closed and all internal NDM buffers are initialized so that subsequent calls to the NDM do not use outdated information.

The **31320 NDM_INITIALIZE** function can be used to shutdown the NDM. When using **31320 NDM_INITIALIZE** to shutdown the NDM, minimum values should be used for all parameters.

For example:

```

9998 GOSUB'31320 NDM_INITIALIZE(0,0,0,1,R)
: IF R<>0 THEN GOSUB 'xxxx(R,"Cannot shutdown NDM")
9999 $END

```

6.5 Creating NDM Data Files

NDM files may be created through either of the following API function calls:

31060 NDM_CREATE_FILE

31070 NDM_CREATE_FILE_FROM_CATALOG

The developer has the choice of using one of these routines to create data files. If the developer has chosen to use the NDM data dictionary files and is using the data description and key description files, the **31070 NDM_CREATE_FILE_FROM_CATA-**

LOG function simplifies coding. However, if the developer has chosen not to use the NDM data dictionary files, then the **31060 NDM_CREATE_FILE** function must be used.

Example 6.2 and Example 6.3 make the following assumptions:

- The file title for the file to be created is CUSTOMER.
- The sample START program described in Section 6.3.7 has been executed.
- The newly created file is not to be used immediately. Therefore, it is closed and must be reopened for use later.
- The sample routines are coded as an internal DEFFN' that can be called from other portions of the application program.

6.5.1 Create File Example 1

Example 6.2 uses the NDM data dictionary files. The data dictionary files used in this example are the NDM catalog, data description and key description.

Example 6.2 - Creating a New File Using 31070

```

5000 DEFFN' 2000(X$,X) :REM X$=FILE TITLE;X=File Number
to use
5010 GOSUB' 31070 NDM_CREATE_FILE_FROM_CATA-
LOG(F$(21),X$,"S",1,F$(X),R)
:REM Catalog file is file number 21 - as set up by
START
:REM OPEN file in shared mode
:IF R=37 THEN xxxx
:REM File already exists - handle this condition dif-
ferently from other return codes that may occur
:IF R<>0 THEN GOSUB'xxxx(R,"Cannot create file")
5020 GOSUB' 31020 NDM_CLOSE_FILE(F$(X),R)
:REM close the file
:IF R<>0 THEN GOSUB'xxxx(R,"Cannot CLOSE file")
5030 RETURN

```

6.5.2 Create File Example 2

In Example 6.3, the catalog file is not used.

Example 6.3 - Creating a New File Without a Catalog File

```

5000 DEFFN' 2000(X$,X) : REM X$=FILE NAME:X=File Han-
dle Number to use
5010 N$(2)="CUSTDD" : REM data description
: N$(3)="CUSTKD" : REM key description file
5020 M$=HEX(0100020A00)
:

```

```

      REM machine specific information based on native ISAM
      : REM used in 31060 call
      : REM hard code machine specific - 512 byte page
      size, 10 preallocation pages
5030 GOSUB '31340 NDM_OPEN_FILE(N$(2),"S",F$(2),R)
      : REM open data description
      : IF R<>0 THEN GOSUB' xxxx (R, " Cannot open data de-
      scription file")
      : REM Always check return codes !!!!
5040 GOSUB '31340 NDM_OPEN_FILE(N$(3),"S",F$(3),R)
      : REM open key description file
      : IF R<>0 THEN GOSUB' xxxx (R, " Cannot open key de-
      scription file")
5050 GOSUB '31290 NDM_GET RE-
CORD_LENGTH_FROM_DD(F$(2),R9,R)
      : REM get record length from data description file
      : IF R<>0 THEN GOSUB' xxxx (R, " Cannot open get re-
      cord length")
5060 GOSUB '31090 NDM_CREATE_KEY_TABLE(F$(2),F$(3),K$,R)
      : REM build key description table
5070 GOSUB '31020 NDM_CLOSE_FILE(F$(2),R)
      : REM close dictionary file
      : IF R<>0 THEN GOSUB' xxxx (R, "Cannot close diction-
      ary file")
      : GOSUB '31020 NDM_CLOSE_FILE(F$(3),R)
      : REM close key description file
      : IF R<>0 THEN GOSUB' xxxx (R, " Cannot close key de-
      scription file")
5080 GOSUB '31060 NDM CRE-
ATE_FILE(X$, "S" 1, K$, R9, M$, F$(X), R)
      : REM create file
      : IF R<>0 THEN GOSUB' xxxx (R, " Cannot create file")
5090 GOSUB '31020 NDM_CLOSE_FILE(F$(X),R)
      : REM close new file
      : IF R<>0 THEN GOSUB' xxxx (R, " Cannot close file")
5100 PRINT "ALL DONE"
5110 RETURN

```

NOTE: The native ISAM code and platform specific information was hard coded on line 5020. Refer to the NDM Platform Specific Addendum for details.

6.5.3 Creating Files with No Data Description or Key Description Files

It is possible to create files without using a data description or key description file. In this case, the application would be responsible for building the key description table itself rather than relying on 31090 NDM_CREATE_KEY_TABLE. To build the key description table, the application would require a method of storing information about the indices of the file to be created. Refer to Appendix B for details on the organization of the key description table.

6.6 Opening a File

The API function to open a file, **31340 NDM_OPEN_FILE**, is very straightforward. If an application is using the NDM features, such as field type conversion and the NDM catalog file, then some additional work is required at the time the file is opened. The example below illustrates how to use these features.

31340 NDM_OPEN_FILE is used to open all files used by a NDM application, whether they are data files or data dictionary files (i.e. data description, key description, catalog, or relation file).

Example 6.4 assumes the following::

- The NDM field type conversion is required.
- The NDM catalog file is to be used.
- The sample START module has been executed.
- The file title to be opened is CUSTOMER.

The example below describes a DEFFN' subroutine that can be called from anywhere in the application. This creates conversion tables for the base record as well as all indices for the specified file. This routine requires that A\$() be dimensioned as a three element array with a length of 72 (A\$(3)72). Q and Q\$ are used by this routine as work variables. Q\$ must be dimensioned to a length of 128.

Example 6.4 - Opening a File

```

5100 DEFFN'2010(X,X1,X$,X1$): REM OPEN Routine
      : REM X=file number
      : REM X1=index number to be set to
      : REM X$=file title
      : REM X1$=MODE ("X" for exclusive or "S" for shared)
5110 GOSUB'31150 NDM_GET_CATALOG_ENTRY(
TRY(F$(21),X$,A$(1),A$(2),A$(3),Q$,R)
      : REM the catalog file is already open as file #21
      : REM the filename, data description name, and key description
name are returned in A$( )
      : IF R<>0 THEN GOSUB'xxxx(R,"Cannot get catalog entry")
5120 GOSUB'31340 NDM_OPEN_FILE(A$(1),X1$,X1,F$(X),R)
      : REM OPEN the data file (A$(1)) as file number X
      : IF R=43 THEN xxx: REM file in use
      : IF R<>0 THEN GOSUB'xxx(R,"Cannot open file")
5130 GOSUB'31340 NDM_OPEN_FILE(A$(2),"S",1,F$(22),R)
      : REM OPEN the data description file (A$(2)) as file number 22
      : IF R=43 THEN xxx: REM file in use
      : IF R<>0 THEN GOSUB'xxx(R,"Cannot open data description File")

```

```

5140 GOSUB'31340 NDM_OPEN_FILE(A$(3),"S",1,F$(23),R)
: REM OPEN the key description file (A$(3)) as file number 23
: IF R=43 THEN xxx: REM file in use
: IF R<>0 THEN GOSUB'xxx(R,"Cannot open key description file")
5150 REM: create the conversion tables for this file
: GOSUB'31050 NDM_CREATE_CONVERSION_TABLE(F$(22),"
",0,C$(X,1),R)
: REM Use element X,1 for the conversion table pointer for the
main record
: REM create the conversion table for all fields defined with
conversion Table number 0 in the data description
5160 REM Now we will create conversion tables for each index
: REM we are assuming that for keyed access, the application
will always pass the key as a concatenated stand alone
string rather than packing the key into the record itself.
Therefore we will set up the conversion table for all indi-
ces to expect this.
: FOR I=1 TO 9: REM Try for all 9 possible indices
: GOSUB'31040 NDM_CREATE_CONV_TA-
BLE_FOR_KEY(F$(22),F$(23),I,C$(X,I+1),"N",R)
: REM use element C$(X,I+1) to store resulting conversion ta-
ble handle
: IF R=115 THEN NEXT I
: REM if a given index is not found, keep going anyway. It is
possible (though unlikely) that a file has been defined with
non-contiguous index numbers.
: IF R<>0 THEN GOSUB'xxxx(R,"Cannot create conversion table
for index")
: NEXT I
5170 REM Now we must close the data description and key descrip-
tion files
: GOSUB'31020 NDM_CLOSE_FILE(F$(22),R): REM data description
: IF R<>0 THEN GOSUB 'xxx(R,"Cannot close data description")
5180 GOSUB'31020 NDM_CLOSE_FILE(F$(23),R): REM key description
file
: IF R<>0 THEN GOSUB 'xxx(R,"Cannot close key description
file")
5190 RETURN

```

This routine can be called from anywhere in the application. For example:

```
0100 GOSUB '2010(1,1,"CUSTOMER","S")
```

This opens the file CUSTOMER as file number 1, setting the current index to 1, in shared mode. For subsequent access to the file CUSTOMER, the application can simply pass F\$(1) to any NDM routine requiring a file handle. In addition, field type conversion tables have been properly established for this file. C\$(1,1) contains the conversion table handle for the base record and C\$(1,1+ I) contains the conversion table handle for each index (I) for this file.

The calling application may be required to check for specific return codes depending on how the application is designed. For example, the '2010 routine may be designed to return to the application if an error code of 43 (File in use), or 11 (File not found) is generated on the actual OPEN call.

6.7 Selecting an Index

Example 6.4 allows specification of the current index to be set when a file is opened. There may be many cases where it is necessary to switch to another index. This can be easily accomplished by use of the **31400 NDM_SET_CURRENT_INDEX** function.

For example, in the CUSTOMER file that has been opened using the sample OPEN routine described above, changing the current index to index number 2 can be done as follows:

```
0100 GOSUB ' 31400 NDM_SET_CURRENT_INDEX(F$(1),2,R)
```

Alternatively, a very simple DEFFN' can be established to perform this task:

```
5200 DEFFN'2020(X,X1): REM set current index;X=file number;
      X1=new index number
5210 GOSUB ' 31400 NDM_SET_CURRENT_INDEX(F$(X),X1,R)
      : IF R<>0 THEN GOSUB'xxx(R,"Cannot set current index")
      : RETURN
```

If this is done, the above application example appears as:

```
0100 GOSUB '2020(1,2)
```

NOTE: If the application needs to restore the original index number, 31180 NDM_GET_CURRENT_INDEX can be used to extract this information and store

the result before changing the index number. Then, this value can be passed back to 31400 NDM_SET_CURRENT_INDEX to restore the original index number.

6.8 Reading a Record by Key

Example 6.5 shows a generalized routine that can be used to read a record by key according to a specified equivalence. The example code assumes that:

- The sample START module has been executed.
- The file being used has been opened by a call to the sample OPEN routine ('2010) described above.
- The index number has been set to the desired value.
- A field type conversion is being used and this is to be performed automatically by the sample routine.

Example 6.5 - Reading by Key

```

5250 DEFFN'2030(X,X1,X2$,X3$,R$): REM read record by key
: REM X=file number
: REM X1=index number
: REM X2$=equivalence type ( =, >, <, <=, >= )
: REM X3$=record lock flag; " " means no lock; "L"
: REM R$ receives the key value. This routine ex-
: REM Note that R$ is the record buffer variable de-
: REM convert the key value
5260 REM convert the key value
: GOSUB'31030 NDM_CONVERT(C$(X,1+X1),R$,1,R)
: REM conversion table handle was established by sam-
: REM convert from Basic-2C to native (direction=1)
: REM The result of this function is that R$ now con-
: REM Now we can perform the READ
5270 REM Now we can perform the READ
: GOSUB'31360 NDM_READ_RE-
: REM File handle was established by sample OPEN
: IF R=1 OR R=2 OR R=6 OR R=42 THEN RETURN
: REM These could be caused by user entry errors. The
: IF R<>0 THEN GOSUB'xxxx(R,"Cannot read by key")
: REM for other return codes, use the general return
5280 REM The actual record is now in R$ - in native for-
mat

```

```

: REM now convert it to Basic-2C Format
: GOSUB'31030 NDM_CONVERT(C$(X,1),R$, -1,R)
: REM Use the conversion table handle for the base re-
:   cord
: REM Convert from native to Basic-2C (direction=-1)
: IF R<>0 THE GOSUB'xxx(R,"Convert of record failed")
: RETURN

```

A typical use of this routine by an application is shown in Example 6.6.

Example 6.6 - Look Up Record by Key

```

0000 REM This routine will extract a specific CUSTOMER
      record based on
      a CUSTOMER NUMBER entered by the operator
0010 DIM K$5
0100 GOSUB'2010(1,1,"CUSTOMER",S): REM open customer
      file
0110 REM get a key value from the operator
      : LINPUT "ENTER CUSTOMER NUMBER "-K$
0120 GOSUB'2030(1,1,"=", " " K$)
      : REM Read record from file 1, current index is num-
      ber 1, get record matching key exactly, do not lock
      the record, key is in K$
      : IF R<>0 THEN DO
      : PRINT HEX(03);
      : IF R=1 OR R=2 OR R=6 THEN PRINT "NOT FOUND"
      : IF R=42 THEN PRINT "Record is locked by other termi-
      nal"
      : GOTO 110
      : ENDDO
0130 REM the record is in R$ in Basic-2C format

```

6.9 Reading Records Sequentially

Records may be read sequentially along a current index, either forward or backward, by use of the **31370 NDM_READ_BY_POSITION** function. However, before **31370 NDM_READ_BY_POSITION** can be used, the current record must be set to a valid value by either a successful **31360 NDM_READ_BY_KEY** call or by a successful **31370 NDM_READ_BY_POSITION** with a position value of -2 (BOF) or + 2 (EOF) specified. If **31370 NDM_READ_BY_POSITION** is used to set the position to BOF, an attempt to read the file backward from that point fails, as will an attempt to read forward from EOF.

The examples in this section (Example 6.7, 6.8, and 6.9) use one generalized routine to perform a **31370 NDM_READ_BY_POSITION** call and then show two different application examples that use this routine. One to read sequential records from BOF, the second to read sequential records starting at a specified key value.

The generalized routine for **31370 NDM_READ_BY_POSITION** assumes that:

- The sample START module has been executed.
- The file being used has been opened by a call to the sample OPEN routine ('2010) described above.
- The index number has been set to the desired value.
- Field type conversions are performed automatically by the sample routine.

Example 6.7 - Reading by Position

```

5300 DEFFN'2040(X,X2,X3$)
      : REM X=file number
      : REM X2=position - may be -2, -1, 0, 1, 2
      : REM X3$=record lock flag; " " means no lock; "L"
      : means lock
      : REM R$ contains the record retrieved
5310 GOSUB'31370 NDM_READ_BY_POSITION(F$(X),R$,X2,X3$,R)
      : REM file handle was established by sample OPEN
      : IF R=1 OR R=2 OR R=42 THEN RETURN
      : REM These could be caused by user entry errors. The
      : calling routine must handle these error codes.
      : IF R<>0 THEN GOSUB'xxxx(R,"Cannot read by position")
      : REM for other errors, use the general error handler
5320 REM The actual record is now in R$ - in native for-
mat
      : REM Now to convert it to Basic-2C Format
      : GOSUB'31030 NDM_CONVERT(C$(X,1),R$,-1,R)
      : REM Use the conversion table handle for the base re-
      : cord
      : REM Convert from native to Basic-2C (direction=-1)
      : IF R<>0 THE GOSUB'xxx(R,"Convert of record failed")
      : RETURN

```

6.9.1 Reading from Logical Start of File

Example 6.8 reads all records from the file CUSTOMER in the logical order according to an index specified by the user.

Example 6.8 - Reading From Start Of File

```

0100 GOSUB'2010(1,1,"CUSTOMER","S") : REM Open file CUS-
TOMER
0110 PRINT "Enter Index Number to use ";
0120 INPUT K
0130 GOSUB '2020(1,K)
      : REM set current index as specified by operator
0140 GOSUB '2040(1,-2," ")
      : REM read the first logical record in the file (posi-
      : tion = -2), do not lock it
0150 IF R=1 THEN 200 : REM - end of file
0160 IF R<>0 THEN GOSUB'XXXX(R,"Cannot Read")
0170 REM Process the record (display it, for example)
0180 GOSUB '2040(1,1," ")
      : REM Read the next logical record (position = 1)
0190 GOTO 150 : REM test return code and process record
0200 REM End of File reached - end of this routine

```

The program segment is structured to have a single routine to test the return code and process the record (lines 150-170) although a record may be returned by any of the three calls to '2040.

6.9.2 Reading Sequentially from a Starting Key Value

Example 6.9 reads all records from the file CUSTOMER in the logical order along index number 1 based on a starting CUSTOMER NUMBER entered by the operator.

Example 6.9 - Reading Sequentially after Positioning

```

0010 DIM K$5
0100 GOSUB '2010(1,1,"CUSTOMER","S") : REM Open file CUS-
TOMER
0110 PRINT "Enter starting customer number to use ";
0120 LINPUT K$
0140 GOSUB '2030(1,1,"=>"," " K$)
      : REM Read the first logical record in the file equal
      or greater than the specified key (K$); do not lock
      it
0150 IF R=1 OR R=6 THEN 200 : REM - EOF
0160 IF R<>0 THEN GOSUB 'XXXX(R, "Cannot Read")
0170 REM Process the record (display it, for example)
0180 GOSUB '2040(1,1," "
      : REM Read the next logical record (position = 1)
0190 GOTO 150: REM test return code and process record
0200 REM EOF reached - end of this routine

```

6.10 Adding New Records

New records can be added to a file by use of the **31330 NDM_INSERT_RECORD** function.

The **31010 NDM_APPEND_UNIQUE_KEY_RECORD** function can also be used to add new records to a file, but this is a specialized API function that should only be used when a sequential key must be generated by the NDM. Most applications will use **31330 NDM_INSERT_RECORD** to add records to a file.

Example 6.10 illustrates a generalized routine to add new records to a file. This routine assumes that:

- The sample START module has been executed.
- The file being used has been opened by a call to the sample OPEN routine ('2010) described above.

- Field type conversion is to be performed.

Example 6.10 - Adding New Records to a File

```

5450 DEFFN'2070(X,X3$): REM Add New Record to file X;
X3$=LOCK CODE
5460 GOSUB'31030 NDM_CONVERT(C$(X,1),R$,1,R)
: REM Conversion Table Handle was established by sam-
: ple OPEN
: REM Convert from Basic-2C to Native (direction=1)
: IF R<>0 THEN GOSUB'xxxx(R,"Convert failed")
5470 GOSUB'31330 NDM_INSERT_RECORD(F$(X),R$,X3$,R)
: IF R=7 THEN RETURN: REM Duplicate Key - application
: must handle this
: IF R<>0 THEN GOSUB'xxx(R,"Cannot Insert record")
: RETURN

```

6.11 Updating Records

Records may be updated by use of the **31380 NDM_REWRITE_RECORD** function. Example 6.11 is a generalized routine for updating records. Example 6.12 is a sample application routine. The generalized routine assumes:

- The sample START module has been executed.
- The file being used has been opened by a call to the sample OPEN routine ('2010) described above.
- Field type conversion is to be performed automatically by the sample routine.
- The current record has been set by a call to **31360 NDM_READ_BY_KEY** or **31370 NDM_READ_BY_POSITION**.
- The record to be rewritten has been locked by the terminal performing the operation. **31380 NDM_REWRITE_RECORD** automatically unlocks the record. This record is stored in R\$.

Example 6.11 - Updating Records

```

5350 DEFFN'2050(X) : REM Rewrite current record in file
number X
5360 GOSUB'31030 NDM_CONVERT(C$(X,1),R$,1,R)
: REM Conversion Table Handle was established by sam-
: ple OPEN
: REM Convert from Basic-2C to native (direction=1)
: IF R<>0 THEN GOSUB'xxxx(R,"Convert failed")
5370 GOSUB'31380 NDM_REWRITE_RECORD(F$(X),R$,R)
: IF R=7 THEN RETURN: REM duplicate key - application
: must handle this
: IF R<>0 THEN GOSUB'xxx(R,"Cannot rewrite record")
: RETURN

```


Example 6.12 - Sample Application Update Routine

```

0000 REM This routine will Extract a specific CUSTOMER
      record based on
      a CUSTOMER NUMBER entered by the operator, allow the
      operator to modify the record and then rewrite it.
0010 DIM K$5
0100 GOSUB'2010(1,1,"CUSTOMER",S): REM Open file CUS-
      TOMER
0110 REM get a key value from the operator
      : LINPUT "ENTER CUSTOMER NUMBER "-K$
0120 GOSUB'2030(1,1,"=", "L", K$ )
      : REM Read Record from file 1, current index is num-
      ber 1, get record matching key exactly, key is in
      K$, lock it
      : IF R<>0 THEN DO
      : PRINT HEX(07);
      : IF R=1 OR R=2 OR R=6 THEN PRINT "NOT FOUND"
      : IF R=42 THEN PRINT "Record is locked by other termi-
      nal"
      : GOTO 110
      : ENDDO
0130 REM the record is in R$ in Basic-2C format
0140 REM Allow the user to modify it
0150 REM Now rewrite modified record
0160 GOSUB'2050(1)
      : IF R=7 THEN DO:
      : REM DUPLICATE KEY - MAKE OPERATOR REENTER OR ABORT
      : ENDDO

```

6.12 Deleting Records

Records may be deleted by use of the **31120 NDM_DELETE_RECORD** function. Example 6.13 is an example of a generalized routine for deleting records. Example 6.14 is a sample application routine. These examples assume:

- The sample START module has been executed
- The file being used has been opened by a call to the sample OPEN routine ('2010) described above.
- The current Record has been set by a call to **31360 NDM_READ_BY_KEY** or **31370 NDM_READ_BY_POSITION**.
- The record to be deleted has been locked by the terminal performing the operation.

Example 6.13 - Sample Code To Show Record Deletion

```

5400 DEFFN'2060(X) : REM Delete current record in file
      number X
5410 GOSUB'31120 NDM_DELETE_RECORD(F$(X),R): REM Delete
      Record

```

```

: IF R<>0 THEN GOSUB'xxx(R,"Cannot Delete record")
: RETURN

```

Example 6.14 - Sample Delete Application Routine

```

0000 REM This routine will extract a specific CUSTOMER
record based on
      a CUSTOMER NUMBER entered by the operator, and then
delete it.
0010 DIM K$5
0100 GOSUB'2010(1,1,"CUSTOMER",S): REM OPEN file CUS-
TOMER
0110 REM GET A KEY VALUE FROM THE OPERATOR
      : LINPUT "ENTER CUSTOMER NUMBER "-K$
0120 GOSUB'2030(1,1,"=", "L", K$ )
      : REM Read Record from file 1, current index is num-
ber 1, get record matching key exactly, key is in
      K$, lock it
      : IF R<>0 THEN DO
      : PRINT HEX(07);
      : IF R=1 OR R=2 OR R=6 THEN PRINT "NOT FOUND"
      : IF R=42 THEN PRINT "Record is locked by other termi-
nal"
      : GOTO 110
      : ENDDO
0130 REM the record is in R$ in Basic-2C format
0150 REM Now delete it
0160 GOSUB'2060(1)

```

CHAPTER 7

SYSTEM AND SUPPORT FILES

7.1 Overview

The NDM Data Dictionary system files are a group of special files used to describe the NDM data dictionary files. The NDM support files are a series of miscellaneous files used by the NDM. These files are all stored as native ISAM data files, and can therefore be read by normal API file access routines. This chapter discusses these files in detail.

Section 7.2 discusses data dictionary system files.

Section 7.3 discusses the NDM support files.

7.2 Data Dictionary System Files

The NDM comes equipped with several data dictionary system files that are used to maintain the NDM data dictionary files. Each type of NDM data dictionary file has two associated data dictionary system files: a data description file and a key description file. There is only one instance of each data dictionary system file on an entire system. For example, one NDMCATK file describes the key structure of every catalog file on every application on a system. These files are shown below.

NDMCAT	Catalog file.
NDMCATD	Data description for the catalog file.
NDMCATK	Key description file for the catalog file.
NDMDDDD	Data description for data description files.
NDMDDKD	Key description for data description files.
NDMKDDD	Data description for key description files.
NDMKDKD	Key description for key description files.
NDMREL	Relation file.
NDMRELD	Data description for the relation file.
NDMRELK	Key description for the relation file.

NOTE: Actual physical file names for these files will vary from one operating system to another.

All data dictionary system files are normally kept in the directory specified by the NDM environment variable. If there is no such variable, the files are kept in directory \NDM (MS-DOS and Novell) or /usr/NDM (Intel and Motorola 68000 XENIX/UNIX).

NOTE: All NDM Utilities provided by Niakwa contain special code to disallow modifying these files.

Use of the data dictionary system files will allow sophisticated applications to perform field type conversion when accessing data description and key description files under program control. For example, if an application needs to access a data description file, it would use NDMDDDD to create the required conversion tables.

7.3 NDM Support Files

There is one instance of each NDM support file, as well as a data description file and a key description file for each, provided with each NDM system.

7.3.1 Field Type File

The field type file contains a record for each field type supported by the native ISAM, as well as a record for each valid Basic-2C field type. These files are:

NDMFTF	List of supported Basic-2C and Native field types.
NDMFTFD	Data description for the field type file.
NDMFTFK	Key description file for the field type file.

Its data description layout is described in Table 7-1.

Start	Length	Type	Field Name
1	1	A	BASIC-2C OR NATIVE
2	1	A	FIELD CLASS
3	2	N	FIELD TYPE NUMBER
5	72	A	DESCRIPTION
77	2	N	MINIMUM LENGTH
79	2	N	MAXIMUM LENGTH
81	2	N	LENGTH INCREMENT
83	1	A	DEFAULT FIELD TYPE (Y/N)
84	141	A	RESERVED FOR NIAKWA EXPANSION

Table 7-1

The field names in Table 7-1 are defined as follows:

BASIC-2C OR NATIVE

This field contains a **B** if this is a Basic-2C field type, or an **N** if it is a native ISAM field type.

FIELD CLASS

This field contains an **A** if this is an alpha field, **N** if numeric, or **D** for date.

FIELD TYPE NUMBER

The number that is used in the data description file to identify this field type.

DESCRIPTION

Narrative description of this field type.

MINIMUM LENGTH

A valid field length for a given type must not be less than the minimum length.

MAXIMUM LENGTH

A valid field length for a given type must not be greater than the maximum length.

LENGTH INCREMENT

A valid field length for a given type must be equal to the minimum length plus some whole multiple of the length increment.

DEFAULT FIELD TYPE

This field contains a **Y** if the field type number specified in this record is returned as the default type for the specified field class when calling **31200 GET_DEFAULT_FIELD_TYPE**. This is meaningful for Native field types only.

This file is different for each native ISAM. The application program may modify this file if desired, but Niakwa **strongly** recommends that any modifications be limited to changing the DEFAULT FIELD TYPE (Y/N) flag.

NOTE: No checking for duplicate DEFAULT FIELD TYPE (Y/N) records is performed.

The key description file for this file is described in Table 7-2.

Index #	Segment #	Ascending	Unique	Field Name
1	0		Y	TYPE NUMBER
1	1	Y		BASIC-2C OR NATIVE
1	2	Y		FIELD TYPE NUMBER
2	0		N	DEFAULT TYPE
2	1	Y		BASIC-2C OR NATIVE
2	2	Y		FIELD CLASS
2	3	N		DEFAULT FIELD TYPE (Y/N)

Table 7-2

7.3.2 Error File

The error file contains a record for each error generated by the NDM functions or the native ISAM. These files include:

NDMERR	List of error messages.
NDMERRD	Data description file for the list of error messages.
NDMERRK	Key description file for the list of error messages.

Its data description layout is defined in Table 7-3.

Start	Length	Type	Field Name
1	2	N	ERROR TYPE
3	2	N	ERROR CODE
5	72	A	ERROR DESCRIPTION
77	20	A	RESERVED FOR NIAKWA EXPANSION

Table 7-3

The field names in Table 7-3 are defined as follows:

ERROR TYPE

The type of return code. 0 for the NDM, or 1 for the native ISAM.

ERROR CODE

A numeric value.

ERROR DESCRIPTION

The text description of the error code.

The key description file for this file is described in Table 7-4.

Index #	Segment #	Ascending	Unique	Field Name
1	0		Y	CODE
1	1	Y		ERROR TYPE
1	2	Y		ERROR CODE

Table 7-4

This file is used by **31210 GET_ERROR_DESCRIPTION**.

The application may use the NDMERR file to store its own error codes and messages. Error types 100 or greater are available for this purpose. Information in the NDMERR may be maintained by the Access Error File Utility - see Chapter 4-6.

7.3.3 Translation Table File

The translation table file contains a record for each type of translation to be done on field type 21 (alpha with translation). Each record consists of two tables, one for translating from Basic-2C to native and another for translating back.

Its data description layout is defined in Table 7-5.

Start	Length	Type	Field Name
1	32	A	TRANSLATION NAME
33	256	A	BASIC-2C TO NATIVE TABLE
289	256	A	NATIVE TO BASIC-2C TABLE
545	64	A	RESERVED FOR NIAKWA EXPANSION

Table 7-5

The field names in Table 7-5 are defined as follows:

TRANSLATION NAME

Name of the translation table.

BASIC-2C TO NATIVE TABLE

Table used to translate Basic-2C characters to native characters.

NATIVE TO BASIC-2C TABLE

Table used to translate native characters to Basic-2C characters.

The key description file for this file is described in Table 7-6.

Index #	Segment #	Ascending	Unique	Field/Index Name
1	0		Y	TRANSLATION NAME
1	1	Y		TRANSLATION NAME

Table 7-6

CHAPTER 8

CONVERTING APPLICATIONS AND DATA FILES

8.1 Overview

This chapter is intended for developers who have applications developed in Basic-2C using record access methods written in Basic-2C.

Because of the lack of a built-in access method in Wang's Basic-2 and Niakwa's Basic-2C prior to the NDM, application developers were forced to devise their own record management systems. Due to the wide variety of record management systems developed in Basic-2/Basic-2C it is not possible to attempt to provide specific details regarding specific applications. Instead, general guidelines for conversion of each broad class of Basic-2/Basic-2C record management system is provided.

General guidelines for conversion of applications data files are also given.

Section 8.2 provides general guidelines for conversion of program code to use the NDM.

Section 8.3 discusses issues relevant to three broad classes of Basic-2/Basic-2C record management systems:

- Systems that provide an indexed access method with generalized subroutines that operate at the record level.
- Systems that provide an indexed access method with generalized subroutines that only work at the key level where actual reading and writing of data is the responsibility of the application modules.
- Systems that rely strictly on DC style record access with no indices.

Section 8.4 provides general guidelines for conversion of existing data files to the NDM.

Section 8.5 provides guidelines for conversion of existing data dictionaries files to the NDM.

8.2 Guidelines for Program Code Conversions

There are several issues the developer must carefully consider when converting existing application code to the NDM. This section attempts to identify some of these issues.

8.2.1 Use of NDM Data Dictionary Files

Although use of the NDM data dictionary files is not required, it is highly recommended. NDM data dictionary files include the catalog, relation, data description and key description files. Refer to Chapter 3 and Chapter 4 for more information.

A catalog entry should be established for each existing data file. The file title should be based on the existing name of the data file. A prefix should also be added to ensure the file titles for the application do not conflict with those used by other applications that may be installed concurrently on the end users' systems.

For each data file, both a data description and key description file should be created. If the application currently uses a data dictionary, this should be converted to the NDM data description and key description files (refer to Section 8.5). If the application does not use a formal data dictionary, the information needed to create the NDM data description and key description files is typically available in the internal documentation of the file layouts. This information should be sufficient to determine the field names, Basic-2C field type, Basic-2C start position, and Basic-2C field length for the data description entries. Determination of native field types in most cases is straightforward.

If the application currently supports indexed access, creation of key description files should be straightforward. All existing indices should be defined. If the application does not use indexed access or if the application has limitations on indexed access that are to be eliminated, careful consideration must be given to the specification of indices. Refer to Section 8.2.4 for more information.

8.2.2 Use of Field Type Conversion

It is essential to use field type conversions if the application uses any field types other than alphanumeric. It is also necessary to use field type conversions if an application wishes to use data from other applications. Refer to Section 5.5 for more information.

There may be cases where the application uses a totally non-standard internal field type to store data. For example, using one byte to contain several bit switches, each of which has a meaning known only to the application. Fields like this can be defined as alphanumeric for both the Basic-2C and native fields. These can even be used as indices if the collating sequence is binary, but these fields will not be understandable by any other program.

NOTE: If it is even remotely possible that these fields are to be used by other programs, then the application must be modified accordingly.

8.2.3 Use of Modular Code

Niakwa strongly recommends using generalized internal subroutines for all calls to the NDM. The reason for this is that multiple NDM calls may be required for a single function and the logic to check return codes should always be used.

For applications that currently use generalized subroutines for data management, implementing this should not be a problem.

For applications that currently use straight line code to access data (typically using DATA LOAD DC/DA), modifying the application structure to use generalized sub-routines for data management is considerably more work initially but often results in reduced maintenance costs and reduced costs for new application development or enhancement.

8.2.4 Use of Indices

The NDM supports up to nine indices per file where each index may contain up to eight segments (there is a maximum of 24 total segments per file). If the application's current data management routines offer less flexibility than this, there may be portions of the application that can be greatly optimized (both in terms of source code management and execution time performance) by using more indices or segments. For example, any portions of the application that search through records for a particular value in a particular field without using an index are good candidates for creating indices.

Another good candidate for indices are applications that use relative record numbers to locate data (typically DC/DA style applications). Although these applications can be converted easily by using the **31010 NDM_APPEND_UNIQUE_KEY_RECORD** function (see Section 8.3.3 for details), the application typically performs much better if the other fields are defined as indices.

NOTE: The amount of work required to use additional indices varies widely from one application to the next. However, if these types of changes are to be made, the best time to make them is when initially converting the application to work with the NDM.

8.2.5 \$OPEN/\$CLOSE

Many Basic-2/Basic-2C applications use \$OPEN to hog a diskimage while multiple operations are performed. Since there is no direct equivalent to \$OPEN/\$CLOSE in the NDM, the developer must consider how \$OPEN/\$CLOSE is being used by the application and modify the application accordingly when converting to the NDM.

Often, the multiple operations performed while a diskimage is locked relate to a single higher level operation such as adding a new record. The key files and the data file are updated and \$OPEN is used to ensure that integrity between the key files and data file is maintained. If this is the only case where \$OPEN/\$CLOSE is used, then the developer can simply disregard the issue since all native ISAMs supported by the NDM ensure the integrity of indices and data.

\$OPEN may also be used to hog a diskimage while multiple high level operations are performed on a single file (for example, the diskimage is \$OPENed while a batch update is performed). Here, the NDM equivalent for \$OPEN is exclusive access to the file. The **31340 NDM_OPEN_FILE** function allows exclusive access to be specified.

If \$OPEN is being used to restrict access to multiple data files concurrently for purposes of a batch update, then the application must take care to avoid potential deadlock conditions when attempting to gain exclusive access to multiple NDM files.

8.2.6 Record Locking

Since there is no language level record locking feature in Basic-2/Basic-2C, developers have devised many creative methods to implement record locking. Some applications use a byte stored with the record to show that a record is locked. Other applications use a separate file to maintain information on locked records. Usually, information about record locking is stored on disk and extra disk I/O is required to lock/unlock records or to check for a record lock.

With the NDM, record locking is performed at the level of the native ISAM and is usually handled via shared memory on multi-user systems. Refer to Chapter 5 in this Programmer's Guide for more information.

When converting applications to the NDM, developers should remove their own record locking logic to avoid degrading performance. Developers should also keep in mind the following about the NDM record locking:

- Record locks can only be set or checked when actually reading a record. Applications that have record locking logic separate from actual I/O may have to be modified to conform to this standard. For optimal performance, the application should avoid performing a read operation solely to lock the record.
- Only one record may be locked per file. Applications that require the ability to lock multiple records within the same file cannot rely solely on the NDM for record locking but must retain at least some of their current record locking mechanisms.
- It is not possible, under the NDM, to learn which terminal has a record locked. All that is known is that the record to be accessed is locked by another task.

8.3 Converting Specific Application Types

This section discusses issues relating to three classes of Basic-2/Basic-2C record management systems: applications using generalized keyed record access, applications using generalized routines to return pointers, and DC style applications.

8.3.1 Applications that use Generalized Keyed Record Access

Applications that use generalized internal subroutines to operate at the record level (the generalized routines read or write the record) and support indexed access typically are the easiest to convert to the NDM.

The general technique for converting these types of applications is to modify the internal **DEFFN**'s to perform appropriate NDM calls instead of managing the data directly. Once modified, the application's internal **DEFFN**' routines should look similar to the examples shown in Chapter 6.

The goal when converting such applications is to leave the actual **DEFFN**' numbers and parameters unchanged so that modifications to the application code that currently call these routines is not modified. However, this goal must be tempered with performance considerations.

For example, when the application has separate **DEFFN**' routines for:

- Fetching a record pointer based on a key value.
- Locking (or unlocking the record).
- Actually reading the record based on the record pointer returned above.

In NDM, the three previous functions are accomplished by the **31360 NDM_READ_BY_KEY** function. The application should not call **31360 NDM_READ_BY_KEY** from each of these routines because this would be reading the record three times when only once is required. Often, this type of condition can be handled without multiple reads.

In the examples shown in Chapter 6, the fetch routine stores the specified key value in a work variable. The lock routine sets a second work variable to show whether a lock is required, and the actual read routine uses these variables in calling **31360 NDM_READ_BY_KEY**. However, in cases where it is not possible to code the generalized routines to handle the situation with a single read, it is probably better to modify the calling routines to perform a single call instead of multiple calls as described above.

8.3.2 Applications that use Generalized Routines to Return Pointers

In this variety of programs, generalized routines are used to provide indexed access to files, but the generalized routines manage the indices and do not actually manage the data. Typically, these routines return a physical sector number for the calling application to use in a DATA LOAD/SAVE BA/DA/DC operation.

For these types of applications, the best technique to convert them to use the NDM is to modify the generalized routines to perform the read, insert, or rewrite operation. The application modules that call these routines have to be modified to skip the actual DATA LOAD/SAVE operations.

8.3.3 DC Style Applications

Applications that strictly use DC statements to manage data with no indexing are typically the most difficult to convert to the NDM, but often have the most to gain.

There are several difficulties with converting DC style applications that the developer must resolve:

- 1) DC style applications do not use a keyed access method. The developer should seriously consider implementing a true keyed access method via the NDM. Where this option is not chosen, the relative record number access of DC style applications can be replicated under the NDM, to a limited extent, by use of the **31010 NDM_APPEND_UNIQUE_KEY_RECORD** function to insert new records.

This function generates a unique sequential key for all new records accessed. This key is the equivalent of a relative record number and may be used to access records just like they are accessed by record number or sector number with a minor calculation by the DC style application.

- 2) DC Style applications sometimes use a list of variables as opposed to a single record buffer variable when reading or writing to disk. All NDM functions are designed to work with a single record buffer variable. Applications that work with a variable list must be modified to pack/unpack the record buffer before and after calls to the NDM. The Basic-2C **\$PACK/\$UNPACK** statement should typically be used for this.
- 3) DC Style applications usually perform record management operations in straight line code as opposed to having generalized routines. This means that every application module that uses the DC statements must be modified.

NOTE: Developers are strongly encouraged to consider implementation of the NDM functions via generalized routines. However, this may be difficult for applications that are not designed to support generalized routines due to line number and variable conflicts.

8.4 Converting Existing Data Files

Conversion of existing data files to the NDM is typically straightforward. First the catalog entry, data description file, and key description file for the file to be created should be established. Then, a data conversion program must be executed. The general outline of a program to convert a Basic-2C data file is shown below:

```

Open Basic-2C Data File]
Point to Basic-2C first record (may not be required)
Create NDM Data File (31070 NDM_CREATE_FILE_FROM_CATALOG)
Create conversion table(31050 NDM_CREATE_CONVERSION_TABLE)
LOOP:Read Next Basic-2C record
    Check for end of file - exit if end of file
    Convert record to Native format(31030 NDM_CONVERT)
    Write Native record(31330 NDM_INSERT_RECORD)
GOTO LOOP
EXIT:Close NDM File(31020 NDM_CLOSE_FILE)
Close Basic-2C File

```

NOTE: The actual logic may vary depending on the methods used by the application's current record management system.

Refer to Chapter 6 for programming examples and suggestions on using the NDM functions referred to in this example.

NOTE: For many applications, it may be possible to develop a simple utility that automatically converts all files.

For DC Style applications, it may be desirable to use **31010 NDM_AP-PEND_UNIQUE_KEY_RECORD** instead of **31330 NDM_INSERT_RECORD**.

8.5 Converting Data Dictionary Files

The NDM data dictionary files are themselves NDM files and may be accessed directly by Basic-2C programs. The organization and layout of the NDM data description and key description files are described in Chapter 3.

Since the Basic-2C data dictionaries themselves can be accessed under program control, it is possible to automate the generation of the NDM data description and key description files.

Factors that the developer must consider when developing a program to convert existing data description files are:

- 1) The NDM catalog, data description and key description files are native ISAM files. Therefore, the field type conversion feature of the NDM must be used.

Data dictionary system files are included with the NDM package. These should be used to create the conversion tables for the catalog, data description and key description files.

The following data dictionary system files should be used:

For catalog files:

NDMCATD is the catalog file definition.
NDMCATK is the key description file definition.

For data description files:

NDMDDDD is the data description file definition.
NDMDDKD is the key description file definition.

For key description files:

NDMKDDD is the data description definition.

NDMKDKD is the key description file definition.

These data dictionary system files reside in the NDM default directory.

- 2) Before creating data description and key description files, entries for each must be placed into the catalog file. The data dictionary system file titles described above should be used in creating the catalog entries. Once the entries for the data description and key description files are created, then **31070 NDM_CREATE_FILE_FROM_CATALOG** can be used, and records can be added to the data description and key description files by **31330 NDM_INSERT_RECORD**. The format for catalog, data description and key description files are described in Chapter 3 of this Programmer's Guide.
- 3) Native field types for each Basic-2C field type used by the application should be determined in advance if possible. Wherever possible, use the default native field types. Access these through **31200 NDM_GET_DEFAULT_FIELD_TYPE**.

CHAPTER 9

NDM EXAMPLE PROGRAMS

9.1 Overview

This chapter describes several NDM example programs that are included on the NDM Development Package diskettes. These programs are intended to provide a working model for developers becoming familiar with NDM. Most common API functions are illustrated as well as several advanced features.

EX1NDM and EX2NDM are two programs that illustrate basic NDM functions such as opening a file, reading records, and the technique for trapping return codes.

The third example TESTCUST is an advanced example that illustrates the recommended programming technique to be used for program development using the NDM.

If the sample programs are modified accidentally, the original programs and supporting files can be copied from the NDM distribution diskettes as often as needed.

NOTE: The sample codes and descriptions of programs' source code presented in this chapter use both the number and name of the API functions. The function name is used for documentation purposes only. Function names should not be used in actual code, only the function numbers.

Section 9.2 discusses starting the example programs provided in this chapter.

Section 9.3 provides two simple examples of Basic-2C programs using the NDM: EX1NDM and EX2NDM.

Section 9.4 discusses the advanced example program: TESTCUST.OBJ.

9.2 Starting the Example Programs

To start the example programs, follow the steps shown below.

1. Start the native ISAM. Refer to the native ISAM's documentation.
2. Get to the directory where the example files are located.
3. Make sure the NDM environment variable is set to the directory where the NDM files are stored. Refer to Chapter 2 of the Platform Specific Addendum for information on setting this variable.
4. Start the Basic-2C RunTime program with the external call feature enabled for the NDM and the example program. For example, using the program TESTCUST on a MS-DOS system:

RTI /XNDMBTRV TESTCUST

or

RTP /XNDMBTRV TESTCUST

Refer to the Platform Specific Addendum for start up instructions for your system.

9.3 Simple Examples Using API Functions

The following examples are intended to illustrate some basic NDM functions such as opening a file and reading records sequentially or by key. These programs do not illustrate more sophisticated NDM functions such as field type conversions.

The examples assume:

- The data file CUSTEXAM has already been created.
- The data file CUSTEXAM has the same file format as the sample CUSTOMER file used throughout this chapter.

The data file used in both of the examples below is identical in format to the data file used in the advanced program example presented in section 9.49

This data file can be accessed by three different indices. These indices were defined when the file was created. The first index will access the file by customer number, the second by customer name, and the third by area code.

9.3.1 NDM Example One (EX1NDM.OBJ)

The first example shows how to read sequentially from the beginning of the file. The code for this example is listed in Example 9-1 and a soft copy is included on the distribution diskettes as EX1NDM.OBJ.

This program reads data sequentially from the "CUSTEXAM" file. Function **31370 NDM_READ_BY_POSITION** begins reading records at the logical beginning of the file, (the first record along index 1). This is accomplished by setting the position parameter to -2 for the first read performed in line 60. All subsequent reads are performed with the position parameter set to 1 to read sequentially forward, (see line 80). A check for EOF, logical end of file, is executed after each read.

Example 9-1

```

0010 REM NDM EXAMPLE 1
0020 DIM C$1024,C1$5,C2$20,C3$13,C4$20,H$2
      : REM C$ = Input Buffer
      : REM C1$ = Customer #
      : REM C2$ = Customer Name
      : REM C3$ = Phone
      : REM C4$ = YTD $
      : REM C5$ = City
      : REM C = total sales (numeric)
0025 DIM O$64: O$=$OPTIONS: STR(O$,22,1)=HEX(17): $OP-
TIONS=O$
      : PRINT HEX(020D0C030E)
      : REM Sets Background/foreground color selection for
      : supported color terminals
0030 PRINT HEX(03);"NDM EXAMPLE #1, LIST CUSTOMER FILE"
      : PRINT HEX(0A)
      : REM Clear Screen, print headings and move down 1
      : line
      : PRINT USING 90,"CUST#","CUSTOMER NAME","PHONE","YTD
      : $","CITY"
      : REM Print column Headings
0040 GOSUB '31320(1,1,20,1024,R)
      : REM Initialize NDM
      : REM Conversion tables = 1
      : REM Conversion table entries = 1
      : REM File count = 20 (total files in application)
      : REM Record Size = 1024 Bytes, thus C$ is dim to 1024
      : IF R<>0 THEN DO
      : PRINT "ERROR ";R;" ON NDM INITIALIZE"
      : STOP #
      : REM Trap error if cannot initialize
      : ENDDO
0050 GOSUB '31340("CUSTEXAM","S",1,H$,R): REM OPEN FILE
      : REM open file (CUSTEXAM)
      : REM file opened in shared mode
      : REM current index set to 1 (by cust #)
      : REM H$ = file handle
      : IF R<>0 THEN DO
      : PRINT "ERROR ";R;" DURING OPEN OF CUSTEXAM FILE"
      : STOP #
      : ENDDO
      : REM Trap error if cannot open file
0060 GOSUB '31370(H$,C$,-2,"",R)
      : REM READ BY POSITION - FROM START OF FILE
      : REM start reading from BOF
      : IF R<>0 THEN DO
      : PRINT "ERROR ";R;" DURING READ BY POSITION TO START
      : OF FILE"
      : STOP #
      : ENDDO
      : REM trap error if problem reading by position
0070 $UNPACK(F=HEX(A005A014A00D52050002A014)) C$ TO
C1$,C2$,C3$,C,C4$
      : PRINT USING 90,C1$,C2$,C3$,C,C4$
      : REM unpack into program variables
0080 GOSUB '31370(H$,C$,1,"",R)
      : REM READ BY POSITION - NEXT RECORD
      : IF R=1 OR R=2 THEN DO : REM END OF FILE
      : PRINT HEX(0A);"END OF FILE REACHED"
      : GOTO 100
      : ENDDO
      : IF R<>0 THEN DO
      : PRINT "ERROR ";R;" DURING READ BY POSITION - NEXT
      : RECORD"
      : STOP #
      : ENDDO

```

```

: REM trap error if cannot read by position
: GOTO 70
: REM continue reading until EOF or BOF (if reading
:   from end of file)
0090 %##### #####
##.##.## #####
0100 GOSUB 31020(H$,R)
: REM Close File
: IF R<>0 THEN DO
: PRINT "ERROR ";R;" DURING CLOSE FILE"
: ENDDO
: REM trap error if cannot close file
0110 END

```

By changing the POSITION parameter in line 60, to a 2, and the POSITION parameter to -1 in line 80, the program would then read the file in reverse, from end to beginning.

9.3.2 NDM Example Two (EX2NDM)

The second example shows how to read the file sequentially starting at a key value selected by the operator. The code for this example is listed in Example 9-2 and a soft copy is included on the distribution diskettes as EX2NDM.OBJ.

The operator can choose to read the file by one of the following three indices: customer number, sort name or sales by area. Once the desired index is input, the program sets the current index to this value by use of NDM function **31400 NDM_SET_CURRENT_INDEX**, (see line 60).

Based on the index, the operator will be prompted for the appropriate values for the key, see lines 80 - 100. Once the key is built and packed, a read by key is performed using the NDM function **31360 NDM_READ_BY_KEY** with search criteria for greater or equal to the input key, (see line 120).

Once the first matching record is found, the remainder of the file is read sequentially using the NDM function **31370 NDM_READ_BY_POSITION**, (see line 160). A check for EOF, logical end of file, is performed.

Example 9-2

```

0010 REM NDM EXAMPLE 2
0020 DIM C$1024,C1$5,C2$20,C3$13,C4$20,H$2
: REM C$ = input buffer
: REM C1$ = customer number
: REM C2$ = customer name
: REM C3$ = phone
: REM C4$ = YTD$
: REM C = total sales (numeric)
0025 DIM O$64: O$=$OPTIONS: STR(O$,22,1)=HEX(17): $OPTIONS=O$
: PRINT HEX(020D0C030E)

```



```

: REM Sets Background/foreground color selection for supported
color terminals
0030 PRINT HEX(03);"NDM EXAMPLE #2, LIST CUSTOMER FILE, BY SE-
LECTED INDEX AND STARTING KEY"
: REM clear screen and print headings
0040 GOSUB '31320(1,1,20,1024,R)
: REM Initialize NDM
: REM Conversion tables = 1
: REM Conversion table entries = 1
: REM File count = 20 (total files in application)
: REM Record Size = 1024 Bytes, thus C$ is dim to 1024
: IF R<>0 THEN DO
: PRINT "ERROR ";R;" ON NDM INITIALIZE"
: STOP #
: ENDDO
0045 GOSUB '31320(2,1,R) : REM SET TOOLBOX STATUS
: IF R <> 0 THEN DO
: PRINT "ERROR ";R;" ON TOOLBOX SET"
: STOP #
: ENDDO
0050 GOSUB '31340("CUSTEXAM","S",1,H$,R): REM OPEN FILE
: REM open file (CUSTEXAM)
: REM file opened in shared mode
: REM current index set to 1 (by cust #)
: REM H$ = file handle name
: IF R<>0 THEN DO
: PRINT "ERROR ";R;" DURING OPEN OF CUSTEXAM FILE"
: STOP #
: ENDDO
0060 PRINT "Enter Index Number to use ";
: INPUT I
: IF I=0 THEN 180
: GOSUB '31400(H$,I,R)
: REM set index based on current file handle H$ and operator
input of index value (I)
: IF R<>0 THEN DO
: PRINT "error ";R;" during Set Index Operation"
: GOTO 60
: ENDDO
: REM trap error if cannot set index
0070 ON I GOTO 80,90,100
: REM branch to appropriate line number based on index input
0080 REM Index 1, reads file by customer number, starting with
value in C1$
: LINPUT "Enter Starting Customer Number ",-C1$
: REM allow operator to select customer number
: IF C1$=" " THEN 60
: REM if null input then allow user to select new index to use
: GOTO 110
: REM go to PACK statement to pack the data into buffer from
the program variables
0090 REM Index 2, reads file by customer name, starting with
value in C2$
: LINPUT "Enter Starting Customer Name ",-STR(C2$,,5)
: REM use the first 5 characters as the key value to speed up
look up
: IF STR(C2$,,5)=" " THEN 60
: REM if null input then allow user to select new index to use
: GOTO 110
: REM go to PACK statement to pack the data into the buffer
from program variables
0100 REM Index 3, reads file by area code, starting with value in
C3$
: LINPUT "Enter Starting Area Code ",-STR(C3$,2,3)
: REM allow operator to select area code
: IF STR(C3$,2,3)=" " THEN 60
: PRINT "Enter Starting Year To Date Sales ";
: INPUT C
: GOTO 110

```

```

: REM go to PACK statement to pack the data into the buffer
: REM from program variables
0110 $PACK(F=HEX(A005A014A00D52050002A014)) C$ FROM
C1$,C2$,C3$,C,C4$
0120 GOSUB '31360(H$,C$,">=", " ",R): REM Read by Key
: REM read by key value from above
: REM H$ = current file handle
: REM C$ = key value to look up, will return record into this
: REM variable
: REM read sequentially starting at input value of C$
: REM " " = don't lock record
: IF R<>0 THEN DO
: PRINT "Error ";R;" during Read by Key"
: GOTO 70
: ENDDO
: REM trap error if cannot read by key
0130 PRINT HEX(0A)
: PRINTUSING 170,"CUST#","CUSTOMER NAME","PHONE","YTD $","CITY"
: REM print blank line and headings
0150 $UNPACK(F=HEX(A005A014A00D52050002A014)) C$ TO
C1$,C2$,C3$,C,C4$
: PRINTUSING 170,C1$,C2$,C3$,C,C4$
: REM unpack input buffer (C$) input program variables
0160 GOSUB '31370(H$,C$,1," ",R)
: REM READ BY POSITION - NEXT RECORD
: IF R=1 OR R=2 THEN DO
: REM END OF FILE OR BEGINNING OF FILE
: PRINT HEX(0A);"END OF FILE REACHED"
: GOTO 60
: ENDDO
: IF R<>0 THEN DO
: PRINT "ERROR ";R;" DURING READ BY POSITION - NEXT RECORD"
: STOP #
: REM trap error if cannot read by position
: ENDDO
: GOTO 150
: REM continue reading until EOF or BOF (if reading from end
: REM of file)
0170 %##### #####
#####
0180 GOSUB '31020(H$,R)
: REM close file
: IF R<>0 THEN DO
: PRINT "ERROR ";R;" DURING CLOSE FILE"
: ENDDO
0190 END

```

9.4 NDM Example Three (TESTCUST)

This advanced example program illustrates API functions, return code trapping, use of modular code, and proper NDM program structure. A soft copy is included on the distribution diskettes as TESTCUST.OBJ.

NOTE: A hard copy of the source code for this program should be printed for easy reference while reviewing this section.

The following sections present the files necessary to run the advanced example program, how to start the program, how to operate the program, the Main Menu functions and finally a detailed discussion of the program code.

NOTE: The following example program is based on the examples, assumptions, and theory presented in Chapter 6 of this Programmer's Guide.

9.4.1 Advanced Program Support Files

The example program is found on the NDM distribution diskettes under the name TESTCUST.OBJ. For the example NDM program to run, several supporting files are necessary. These files are also found on the NDM distribution diskettes with the file names shown below:

CUSTCAT	The example program's catalog file.
CUSTDD	The example program's data description file.
CUSTKD	The example program's key description file.
CUSTREL	The example program's relation file.

All these files may be viewed and modified by using the API functions or NDM Utilities. Refer to Chapter 4 for operating instructions for these utilities.

9.4.2 Advanced Program General Operation

Once the Basic-2C RunTime program is started, Basic-2C runs its normal security check procedure, displays the licensing information, checks the NDM security, and executes the example program. When this is completed, the example program's Main Menu appears as shown in Figure 9-1 below.

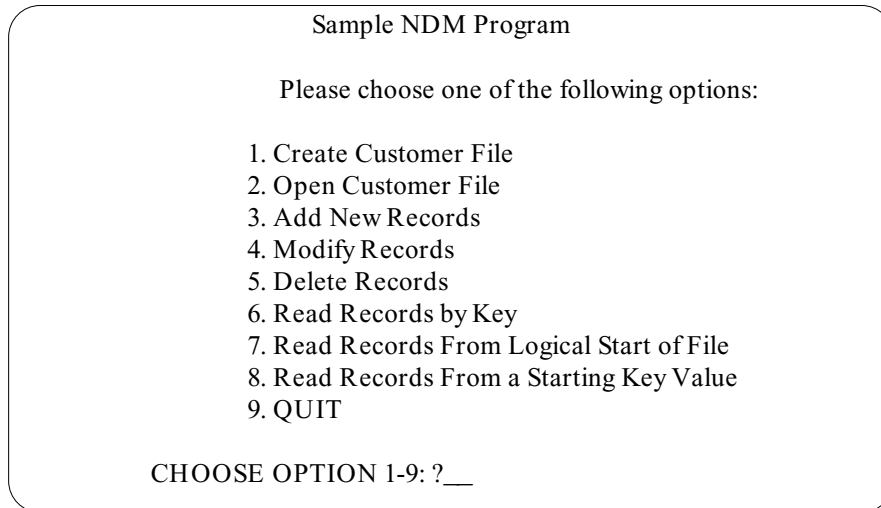


Figure 9 - 1

To choose a menu selection, enter the number of the option to run and press Enter.

9.4.3 Create Customer File

This option allows the operator to create the customer file CUSTOMER. If the file CUSTOMER already exists, the message "FILE ALREADY EXISTS - HIT ANY KEY TO CONTINUE" appears.

NOTE: The customer file MUST exist before any other example program options can be used.

9.4.4 Open Customer File

This option allows the operator to open the CUSTOMER file. If the file is opened, the message "The Customer file was opened successfully. Press any key to continue:" is displayed.

9.4.6 Modify Records

This option operates the same as the ADD NEW RECORDS option. The Modify Screen begins by displaying the prompt "PLEASE ENTER CUSTOMER NUMBER (BLANK TO QUIT):".

NOTE: The CUSTOMER NUMBER must be an existing customer number, otherwise the 'RECORD NOT FOUND' message is displayed.

If a valid record was entered, all existing data is displayed giving the operator the ability to modify each entry as it appears. Once the information is modified, the example program displays the message "MODIFY SUCCESSFUL" when finished and reissues the "PLEASE ENTER CUSTOMER NUMBER (BLANK TO QUIT):" prompt.

9.4.7 Delete Records

This option allows the operator to delete an existing customer record. When this option is run, the operator is prompted to "PLEASE ENTER CUSTOMER NUMBER (BLANK TO QUIT):". If a valid customer number is entered, the message "DELETE SUCCESSFUL" appears and the "PLEASE ENTER CUSTOMER NUMBER (BLANK TO QUIT):" prompt is reissued.

To delete another record, enter the record number or press ENTER to return to the Main Menu.

NOTE: If an invalid record number is entered, the message 'RECORD NOT FOUND' is generated by the example program.

9.4.8 Read Records by Key

This option allows the operator to display each record by key. When this option is selected, the sample program displays the prompt "PLEASE ENTER CUSTOMER NUMBER (BLANK TO QUIT):". If a valid customer number is entered, the sample program displays the record's information as in the example shown below in Figure 9-2.

```
CUSTOMER NUMBER           : 12345
CUSTOMER NAME : John Doe
CUSTOMER PHONE : (303)443-1234
YTD SALE$$ : 375000
NUMBER OF SALES : 37
CUSTOMER CITY : Boulder
```

Figure 9 - 2

Once this information is displayed, the "PLEASE ENTER CUSTOMER NUMBER (BLANK TO QUIT):" prompt is reissued. The operator may then enter another valid customer number or press Enter to return to the Main Menu.

NOTE: If an invalid customer number is entered, the sample program displays the message 'RECORD NOT FOUND' and reissues the 'PLEASE ENTER CUSTOMER NUMBER (BLANK TO QUIT):' prompt.

9.4.9 Read Records From Logical Start of File

This option displays a listing of the customer data file starting from the logical beginning of file using whatever index number was entered at the "PLEASE ENTER INDEX NUMBER TO USE : ?" prompt. Index number 1 corresponds to the CUST # index that contains the CUSTOMER NUMBER as key, index 2 corresponds to the SORT NAME index that contains NAME-I as key and index number 3 corresponds to SALES BY AREA index that contains the AREA CODE and YTD SALE\$\$ as keys.

An example of the output from this option is shown below in Table 9-1 sorted by index 2, Name.

CUST#	NAME	PHONE	SALE\$	# SALES	CITY
34567	Bill Ross	(208)679-0708	23000	6	Boise
45678	Eric Jones	(708)985-3343	427000	44	Chicago
12345	John Doe	(303)443-1234	375000	37	Boulder
23456	Karen Smith	(619)883-5555	50000	12	Eugene

Table 9-1

9.4.10 Read Records From a Starting Key Value

This option displays a listing of the customer data file starting from the customer number entered at the "PLEASE ENTER STARTING CUSTOMER NUMBER TO USE : " prompt.

An example of the output from this option is shown below in Table 9-2 starting from customer number 12345.

CUST #	NAME	PHONE	SALES	# SALES	CITY
12345	John Doe	(303)443-1234	375000	37	Boulder
23456	Karen Smith	(619)883-5555	50000	12	Eugene
34567	Bill Ross	(208)679-0708	23000	6	Boise
45678	Eric Jones	(708)985-3343	427000	44	Chicago

Table 9-2

9.4.11 QUIT

This option exits the NDM sample program and returns the operator to a Basic-2C prompt. From here, any standard Basic-2C statement may be issued. This allows the operator to examine, print, and modify the sample program source code as needed.

9.4.12 Advanced Program Main Features

The following is a list of the main NDM programming features that this sample program attempts to illustrate. These features include:

- Proper NDM program structure.
- Use of modular code.
- How to check for return codes.
- The basic API functions:

- Initialize NDM
- Create a file
- Open a file
- Field level conversions
- Set current index
- Read a record
- Delete a record
- Write a record
- Update a record
- Close a file.

The remainder of this section discusses these key features from a programming development perspective -- specifically how the actual programming is done for these features.

9.4.13 Advanced Program Operation

This example program's operation clearly illustrates how a NDM based Basic-2C program may be used for a simple data base type application. This example was selected based on its use of many essential API functions that are necessary for most applications that would use the NDM. The following section examines the Basic-2C source code and the API function calls used to explain how these main features are coded in a typical Basic-2C/NDM application.

Line	Routine Purpose	Calls to / Description
65	Initialize	GOSUB '3000 (line 3000)/Initialize NDM, Open Catalog
71	Main Menu	- / Choose option to perform
80	Create file CUSTOMER	GOSUB '2000 (line 2000)/Create file "CUSTOMER", handle 1 Check return code 37 (file already exists) - print error messages. Return to Main Menu (line 71)
120	Open File Customer	GOSUB '3500 (line 3500)/Open file"CUSTOMER",in shared mode, set index to index 1, file handle is 1. Print successful message. Return to Main Menu (line 71).
200	Add New Records	GOSUB '4000 (line 4000)/Input Records Routine Return to Main Menu (line 71).
400	Modify Records	GOSUB '6800 (line 6800)/Prompt, Read and lock desired record. GOSUB '4500 (line 4500)/UnPack record GOSUB '5500 (line 5500)/ReWrite Record Check for duplicate record Modify Successful Message Goto 400 for more records to modify.
600	Delete Records	GOSUB '6800 (line 6800)/Prompt, Read and lock desired record. GOSUB '5800 (line 5800)/Delete desired record. Delete Successful Message Goto 600 for more records to delete.
800	Read Records by Key	GOSUB '2500 (line 2500)/Set Current Index to Index 1 GOSUB '6800 (line 6800)/Prompt and Read desired record. GOSUB '4500 (line 4500)/UnPack record Display the Record (line 860) Goto 820 for more records to display

Line	Routine Purpose	Calls to / Description
1000	Read records from start of file	Get Index Number from the operator Check for valid index GOSUB '2500 (line 2500)/Set Current Index to Desired Index GOSUB '6000 (line 6000)/Read first record in file 1, position -2, no locking
1050		Check for EOF, if reached goto 1000 for a new index Check if record was blocked by other user/operation GOSUB '4500 (line 4500)/UnPack record Display the Record (line 1090) GOSUB '6000 (line 6000)/Read next in file 1, record position 1, no locking Goto 1050
1300	Read records from a start key	GOSUB '2500 (line 2500)/Set Current Index to Index 1 Get starting key value GOSUB '6800 (line 6800)/Prompt and Read desired record, do not lock records, read record equal or greater than the key.
1360		Check for EOF, if reached goto 1450 for a new key Check if record was blocked by other user/operation GOSUB '6000 (line 6000)/Read record in file 1, current position, 0, no locking
1050		GOSUB '4500 (line 4500)/UnPack record Display the Record (line 1090) GOSUB '6000 (line 6000)/Read next record in file 1, record position 1, no locking Goto 1350
2000	DEFFN '2000(N\$,N)	Create file N\$ using NDM Catalog, DD, KD file GOSUB '31070 Create_File_From_Catalog Check return code (if file already exists and/or other errors) GOSUB '31020 NDM_CLOSE_FILE Return

Line	Routine Purpose	Calls to / Description
2500	DEFFN '2500(N,I1)	Set current index in File Number N to I1 GOSUB '31400 NDM_SET_CURRENT_INDEX Check return code Return
2800	Prompt & Pack Routine	Display all fields of records and allow for input. GOSUB '4500 (line 4500)/Pack the record read. Return
3000	DEFFN '3000(N\$)	Initialize Routine GOSUB '31320 NDM_INITIALIZE_ROUTINE Check return code GOSUB '31420 NDM_SET_TOOLBOX_STATUS Check return code GOSUB '31340 NDM_OPEN_FILE for catalog N\$ Check return code GOSUB '31160 NDM_GET_CONFIGURATION Check Return Code Return

Line	Routine Purpose	Calls to / Description
3500	DEFFN '3500(A\$,M1\$.I1,H)	<p>Open file A\$, in mode M1\$, Set Current index to I1, the open file will have handle H.</p> <p>GOSUB '31150</p> <p>NDM_GET_CATALOG_ENTRY to get the DD and KD files for file A\$</p> <p>Check return code</p> <p>GOSUB '31340 NDM_OPEN_FILE for file A\$, in mode M1\$, set index to I1, file handle H.</p> <p>Check return code</p> <p>GOSUB '31340 NDM_OPEN_FILE for DD file, in mode M1\$, set index to 1, file handle 22.</p> <p>Check return code</p> <p>GOSUB '31340 NDM_OPEN_FILE for KD file, in mode M1\$, set index to 1, file handle 23.</p> <p>Check return code</p> <p>GOSUB '31050</p> <p>NDM_CREATE_CONVERSION_TABLE for the main record, using element H,1 for the conversion table pointer, all fields defined as main record file are included.</p> <p>Check return code</p> <p>GOSUB '31040</p> <p>NDM_CREATE_CONV_TABLE_FOR_KEY for the other 2 possible indices, using element H,I+ 1 for the conversion table pointer, all fields defined as main record file are included.</p> <p>Check return code</p> <p>GOSUB '8500 (Line 8500)/Close DD file</p> <p>GOSUB '8500 (Line 8500)/Close KD file</p> <p>Return</p>

Line	Routine Purpose	Calls to / Description
4000	DEFFN '4000	Input New Records GOSUB '2800/Prompt and Pack New Record GOSUB '31030 NDM_CONVERT the whole record stored in R\$ Check return code GOSUB '31330 NDM_INSERT_RECORD R\$ but do not lock Check return code (for duplicate records or other errors) Goto 4010 for more records to add Return
4500	DEFFN '4500(Z2\$)	Pack/UnPack for data file in function if Z2\$= "P"/"U" Return
5000	DEFFN '5000(N,I1,O9\$,L1,R\$)	Read record by key in file number N, with current index set to I1, equivalence type O9\$, Record Lock Flag L1\$, the key value will be received in R\$ starting in byte 1. GOSUB '31030 NDM_CONVERT from Basic-2C to Native, R\$ will contain the key value in native format in record position. Check return code GOSUB '31360 NDM_READ_RECORD_BY_KEY Check return code (for error code 2, 6, 12 or other errors) GOSUB '31030 NDM_CONVERT from native to Basic-2C, using conversion table for basic record. Check return code Return

Line	Routine Purpose	Calls to / Description
5500	DEFFN '5500(N)	Rewrite record in file number N GOSUB '31030 NDM_CONVERT from Basic-2C to native, use conversion table handle for main record. Check return record. GOSUB '31380 NDM_REWRITE_RECORD Check return code (for duplicate records or other records). Return
5800	DEFFN '5800(N)	Delete current record in file number N GOSUB '31120 NDM_DELETE_RECORD Check return code Return
6000	DEFFN '6000(N,P2,L1\$)	Read Record in file number N, by position P2, with Record Lock Flag L1\$ GOSUB '31370 NDM_READ_RECORD_BY_POSITION Check return code (error code 1, 2 or 42, or other error codes) GOSUB '31030 NDM_CONVERT from native to Basic-2C, using conversion table for basic record. Check return code Return
6800	DEFFN '6800(L1\$,O9\$)	Prompt and Read Input of the Customer Number GOSUB '5000 (Line 5000) - Read record from file 1, set current index to 1, match key value exactly, lock the record. Check return code for values of 1,2 or 6 (Record not found), or 42 (Record locked by other user). Goto 6800 Return
8000	Shutdown routine	GOSUB '31320 NDM_INITIALIZE with parameters of 0,0,1,1,R to close down.

Line	Routine Purpose	Calls to / Description
8500	DEFFN '8500(H)	Close file with handle H GOSUB '31020 NDM_CLOSE_FILE Check return code Return
9000	DEFFN '9999(R,D1\$)	Error Routine to display error message D1\$ plus the corresponding error message for error code R. GOSUB '31210 NDM_GET_ERROR_DESCRIPTION Check return code Print D1\$,R,Error Description. Return

CHAPTER 10

API FUNCTIONS

10.1 Overview

This chapter describes all the NDM function calls in numeric order.

Section 10.2 provides a reference list of all NDM API functions (31010 - 31460).

Section 10.2 describes each API function including a description of each input and output parameter, a detailed description of the function, any pre-requisites required before using the function, a discussion of the currency changes caused by the function, a description of toolbox feature considerations, a list of the possible errors that may be generated by the function call, and a list of other functions to refer to for additional information.

NOTE: The possible errors section listed for each function includes only the errors that are likely to occur under normal operation.

NOTE: Examples in this chapter use both the number and name of the API functions. The API function name is used for documentation purposes only. API function names should not be used in actual code, only the API function numbers. Long variable names for the parameters are for documentation purposes only. When actually programming, use the standard Basic-2C variable formats.

Numeric parameters of these API functions must have integer values - passing a non-integer value will result in the signalling of a Basic-2C return number 303.

10.2 API Function Calls

The following is a list of the available NDM API function calls:

```
31010  NDM_APPEND_UNIQUE_KEY_RECORD
31020  NDM_CLOSE_FILE
31030  NDM_CONVERT
31040  NDM_CREATE_CONV_TABLE_FOR_KEY
31050  NDM_CREATE_CONVERSION_TABLE
31060  NDM_CREATE_FILE
31070  NDM_CREATE_FILE_FROM_CATALOG
31080  NDM_CREATE_INDEX
31090  NDM_CREATE_KEY_TABLE
31100TNDM_DELETE_FILE
31110  NDM_DELETE_INDEX
31120  NDM_DELETE_RECORD
31130  NDM_DESTROY_CONVERSION_TABLE
31140  NDM_GET_ISAM_ERROR_CODE
31150  NDM_GET_CATALOG_ENTRY
31160  NDM_GET_CONFIGURATION
31170  NDM_GET_CONVERSION_TABLE_LIST
31180  NDM_GET_CURRENT_INDEX
31200  NDM_GET_DEFAULT_FIELD_TYPE
31210  NDM_GET_ERROR_DESCRIPTION
31220  NDM_GET_FILE_STATUS
31223  NDM_GET_FORMAT_SPEC_FOR_KEY
31225  NDM_GET_FORMAT_SPEC
31230  NDM_GET_HANDLE_SIZE
31240  NDM_GET_INDEX_NUMBER
31250  NDM_GET_KEY_TABLE_SIZE
```

31260 NDM_GET_LIMIT_HIGHWATERS
31270 NDM_GET_OPEN_FILE_LIST
31280 NDM_GET_POSITION
31290 NDM_GET_RECORD_LENGTH_FROM_DD
31310 NDM_GET_TOOLBOX_STATUS
31315 NDM_GET_TRANSLATION_TABLE
31320 NDM_INITIALIZE
31330 NDM_INSERT_RECORD
31340 NDM_OPEN_FILE
31360 NDM_READ_RECORD_BY_KEY
31370 NDM_READ_RECORD_BY_POSITION
31380 NDM_REWRITE_RECORD
31400 NDM_SET_CURRENT_INDEX
31410 NDM_SET_POSITION
31420 NDM_SET_TOOLBOX_STATUS
31425 NDM_SET_TRANSLATION_TABLE
31430 NDM_TRANSACTION_ABORT
31440 NDM_TRANSACTION_COMPLETE
31450 NDM_TRANSACTION_START
31460 NDM_UNLOCK_ALL_RECORDS

10.3 API Function Listing in Numeric Order

The following is a detailed description of each of the NDM API functions. These are listed in numeric order for easy reference.

```
' 31010 NDM_APPEND_UNIQUE_KEY_RECORD
      (FILE_HANDLE$, RECORD_BUFFER$, LOCK$, RETURN_CODE)
```

Input Parameters:

FILE_HANDLE\$ - the handle to the open file that will receive the new record.

RECORD_BUFFER\$ - the record to be inserted, in native ISAM format, less the current index's key.

LOCK\$ - contains one of the following values:

"L" lock this record if it was written successfully.

" " don't lock this record.

Output Parameters:

RECORD_BUFFER\$ - the portion of this buffer corresponding to the current index's key will contain the newly created key value.

Description:

This will add the specified record to the specified file, giving it a unique key in the current index.

NOTE: The native ISAM is asked directly by the API as to key position, length, and type for the current index.

The current index's key must have only one segment and will not allow duplicates.

The data in RECORD_BUFFER\$ must be in "native" format - this means that to write a Basic-2C record, the API function **31030 NDM_CONVERT** must be called first.

This function is intended to ease conversion of applications that use DC-style data management with no actual key. The sequential key generated can be used as a key for "seek" operations. Performing seeks against this key will be roughly equivalent to performing DSKIP/DBACKSPACE operations using the record number option. For DC-style applications that use the sector number option of DSKIP/DBACKSPACE, the equivalent key to seek can be calculated by dividing the "sector number" by the number of sectors per record.

Applications that do not require generation of a unique sequential key should use **31330 NDM_INSERT_RECORD** to add new records to an NDM file.

Locked records may be unlocked by calling either **31380 NDM_REWRITE_RECORD** or **31460 NDM_UNLOCK_ALL_RECORDS**.

Pre-Requisites:

- The FILE_HANDLE\$ file must be open.
- The current index of this file must be set to an index whose key has a single segment and does not allow duplicates.
- The RECORD_BUFFER\$ is loaded with valid native data in all fields except the key field of the current index. This is usually done by calling **31030 NDM_CONVERT** with a special conversion table that converts all fields except the key.

Currency Changes:

The current record of the file is set to the newly created record.

Toolbox Features:

None

Possible Errors:

40 lost position. The file's last record was deleted, or its key was modified, during this operation.

See Also:

31030 NDM_CONVERT
31330 NDM_INSERT_RECORD
31380 NDM_REWRITE_RECORD

' 31020 NDM_CLOSE_FILE (FILE_HANDLE\$, RETURN_CODE)
--

Input Parameters:

FILE_HANDLE\$ - the handle to the file to be closed.

Output Parameters:

None

Description:

This function closes the specified open file. If the file has a locked record, the lock will be released.

A file may not be closed inside a transaction.

Pre-Requisites:

- The FILE_HANDLE\$ file must be open.

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

None

See Also:

31340 NDM_OPEN_FILE

```
' 31030 NDM_CONVERT  
      (CONVERSION_TABLE$, RECORD_BUFFER$, DIRECTION, RETURN_CODE)
```

Input Parameters:

CONVERSION_TABLE\$ - a two-byte conversion table handle.

RECORD_BUFFER\$ - the data to be converted, in either Basic-2C or native ISAM format.

DIRECTION - contains one of:

- 1 data is converted from Basic-2C format to native ISAM format.
- 1 data is converted from native ISAM format to Basic-2C format.

Output Parameters:

RECORD_BUFFER\$ - the converted data record, in either native ISAM or Basic-2C format.

Description:

This Function converts the data in RECORD_BUFFER\$ according to the information in the specified conversion table. If this table does not describe every byte of the source record, the undefined bytes will contain undefined (and possibly invalid) values. Therefore, if you plan to read a record, update a single field, and write it back, you must define the bytes you are not interested in with the conversion table (their type can be set to "string" so that they will be copied straight across).

Failure to do so will result in data being lost.

Pre-Requisites:

- 31050 NDM_CREATE_CONVERSION_TABLE or 31040 NDM_CREATE_CONV_TABLE_FOR_KEY must have been called to create the CONVERSION_TABLE\$.

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

142 Numeric overflow during conversion

143 Numeric underflow during conversion

See Also:

31040 NDM_CREATE_CONV_TABLE_FOR_KEY

31050 NDM_CREATE_CONVERSION_TABLE

```
' 31040 NDM_CREATE_CONV_TABLE_FOR_KEY
      (DD_HANDLE$, KEYDESC_HANDLE$, INDEX_NUMBER,
      CONVERSION_TABLE$, KEY_IN_RECORD$, RETURN_CODE)
```

Input Parameters:

DD_HANDLE\$ - the handle to the open data description file containing conversion information for the desired fields.

KEYDESC_HANDLE\$ - the handle to the open key description file containing a description of the key to create a conversion table for.

INDEX_NUMBER - the number to match the "INDEX NUMBER" field of the key description record.

KEY_IN_RECORD\$ - contains "Y" if the keys to be converted by this conversion table will be in their correct places in the Basic-2C record, or "N" if they will be contiguous at the start of the record.

Output Parameters:

CONVERSION_TABLE\$ - a two-byte handle to the newly created conversion table.

Description:

This function creates a conversion table describing the key segments of a particular index of a file. This table can then be used to convert keys before performing key operations such as **31360 NDM_READ_BY_KEY**.

The specified key description file is searched for entries with the specified INDEX_NUMBER. For each of these, the entry in the data description file with a FIELD NAME matching the key description file's entry supplies the conversion information for that field.

If the keys to be used will be placed into their normal positions in a Basic-2C record, then a "Y" should be passed in KEY_IN_RECORD\$. If a "N" is passed in this value, keys must be placed contiguously into the Basic-2C record starting at byte 1, in order of segment number.

This routine needs to be called only once, at the start of a program, for each conversion table required.

Pre-Requisites:

- The key description and data description files must be opened.

Currency Changes:

The current record and current index of both the key description and data description files are changed.

Toolbox Features:

None

Possible Errors:

- 40 Lost position. The current record of the key description or data description file was deleted, or its key was modified.
- 42 A required record in the key description file or the data description file is in use.
- 43 Either the key description file or the data description file is in use.

See Also:

- 31050 NDM_CREATE_CONVERSION_TABLE
- 31360 NDM_READ_RECORD_BY_KEY

```
' 31050 NDM_CREATE_CONVERSION_TABLE
      (DD_HANDLE$, FIELD_NAMES$, CONVERSION_TABLE_NUMBER,
       CONVERSION_TABLE$, RETURN_CODE)
```

Input Parameters:

DD_HANDLE\$ - the handle to the open data description file describing the data record to be converted.

FIELD_NAMES\$ - a list of field names, as stored in the data description, to be used in the conversion table. Multiple names are separated by commas. If this parameter is blank, all fields with the specified conversion table number will be used.

CONVERSION_TABLE_NUMBER - the number of the conversion table to create. This value is matched against the "conversion table number" field of the data description records. If names are specified in FIELD_NAME\$, this value is ignored.

Output Parameters:

CONVERSION_TABLE\$ - a two-byte handle to the newly created conversion table.

Description:

This function reads the conversion information in the specified data description file into an internal conversion table and returns a handle to it. Only records in the data description file whose "conversion table number" field matches the CONVERSION_TABLE_NUMBER parameter, or whose names are specified in FIELD_NAMES\$ (if any) are stored in the conversion table. The resulting CONVERSION_TABLE\$ handle can then be used in subsequent **31030 NDM_CONVERT** calls.

This routine need only be called once, at the start of a program, for each conversion table needed.

Pre-Requisites:

- The data description file must be open.

Currency Changes:

The current record of the data description file is changed.

Toolbox Features:

None

Possible Errors:

40 Lost position. The current record of the data description file was deleted, or its key was modified.

42 A required data description record is in use.

See Also:

31030 NDM_CONVERT

```
31060 NDM_CREATE_FILE
      (FILE_NAME$, MODE$, INDEX_NUMBER, KEYDESC_TABLE$,
       RECORD_LENGTH, ISAM_SPECIFIC$, FILE_HANDLE$, RETURN_CODE)
```

Input Parameters:

FILE_NAME\$ - the name of the file to be created. This must not include an extension (unless the "filename extensions" toolbox feature is enabled).

MODE\$ - one character:

- S open in shared mode
- X open in exclusive mode
- R open in read only mode

INDEX_NUMBER - the value to set the current index of the new file to.

KEYDESC_TABLE\$ - describes the key structure of the file to be created. Usually created with **31090 NDM_CREATE_KEY_TABLE**.

RECORD_LENGTH - the length in bytes of a record in the file to be created. This can be determined by a call to **31290 NDM_GET_RECORD_LENGTH_FROM_DD**.

ISAM_SPECIFIC\$ - a 128-byte string containing native ISAM specific data used in file creation. Usually created with **31150 NDM_GET_CATALOG_ENTRY**.

Output Parameters:

FILE_HANDLE\$ - a handle to the newly created file, opened in the specified mode.

Description:

This function creates a new data file with the specified name, key description, and record length, and opens this file in the specified mode. This function does not create a catalog file entry for the new file - this must be done in a separate call.

The data file created by this call is left open in the mode specified in MODE\$ (see the description of **31340 NDM_OPEN_FILE**) and the current index is set to the given value. A handle to the file is returned in FILE_HANDLE\$, which should be at least as big as the size returned by a previous call to **31230 NDM_GET_HANDLE_SIZE**.

The current record of the newly created file is set to "undefined".

The current index of the new file is set to the value specified in INDEX_NUMBER. If this value is zero, the current index is undefined. If the specified index is invalid (for example, unsupported native ISAM specific features are in use), the current index is undefined.

The index numbers supplied in the KEYDESC_TABLE\$ parameter are used by the NDM only to determine the boundaries between the different keys. This means that if the indices are not numbered consecutively, they will be implicitly renumbered by the NDM.

For example, if the KEYDESC_TABLE\$ (which is usually created from the data description file created by the NDM Utilities) specifies index numbers 1, 1, 3, 3, 3, 5, 0 for its segments, the file created will have a two-segment index as index number 1, a three-segment index as index number 2, and a single-segment index as index number 3. This may cause confusion when running the NDM Utilities if the expected index numbers were 1, 3, 5.

Pre-Requisites:

- Optional - Open the catalog file and call **31150 NDM_GET_CATALOG_ENTRY** to get ISAM_SPECIFIC\$ and the name of the key description file.
- Optional - Open the key description file and data description file.
- Optional - Call **31090 NDM_CREATE_KEY_TABLE** to create the KEYDESC_TABLE\$ variable.

Currency Changes:

The current index of the new file is set to the value passed in INDEX_NUMBER (unless it's zero or the index is invalid). The current record is set to "invalid".

Toolbox Features:

If the "filename extensions" feature is enabled (feature # 1), extensions may be specified in the FILENAME\$ parameter. To specify a name with no extension, and to avoid the addition of a default extension, add a period at the end of the name.

If any index segment is an extended key type, it will be necessary to enable the "extended key type feature" whenever using that index.

If the "file limits" feature is enabled (feature # 3) then the NDM limits are replaced by the native ISAM's limits (e.g., depending on the native ISAM, a key longer than 120 bytes may be defined).

Possible Errors:

- 10 Invalid file name.
- 37 The file already exists.
- 48 Permission error. Network privileges disallow file creation.
- 115 The specified filename contains an extension, but the "filename extensions" tool-box feature has not been enabled.

See Also:

- 31090 NDM_CREATE_KEY_TABLE
- 31100 NDM_DELETE_FILE
- 31150 NDM_GET_CATALOG_ENTRY
- 31230 NDM_GET_HANDLE_SIZE
- 31290 NDM_GET_RECORD_LENGTH_FROM_DD
- 31340 NDM_OPEN_FILE


```
' 31070 NDM_CREATE_FILE_FROM_CATALOG
      (CATALOG_HANDLE$, FILE_TITLE$, MODE$, INDEX_NUMBER,
       FILE_HANDLE$, RETURN_CODE)
```

Input Parameters:

CATALOG_HANDLE\$ - the handle to the open catalog file containing the entry for the file to be created.

FILE_TITLE\$ - a 32-character mnemonic name for the file, which is matched against the "file title" field of the catalog file.

MODE\$ - one character:

- S open in shared mode
- X open in exclusive mode
- R open in read only mode

INDEX_NUMBER - the value to set the current index of the new file to.

Output Parameters:

FILE_HANDLE\$ - a handle to the newly created file, opened in the specified mode.

Description:

This function creates a new file from its entry in a catalog file. Calling this routine is equivalent to the following calls:

```
31150 NDM_GET_CATALOG_ENTRY
31340 NDM_OPEN_FILE
31340 NDM_OPEN_FILE
31090 NDM_CREATE_KEY_TABLE
31290 NDM_GET_RECORD_LENGTH_FROM_DD
31020 NDM_CLOSE_FILE
31020 NDM_CLOSE_FILE
31060 NDM_CREATE_FILE
```

If any of the above calls returns a non-zero RETURN_CODE, execution terminates and the code is returned in the RETURN_CODE for this call.

As with **31060 NDM_CREATE_FILE**, the current record of the newly created file is "undefined", and passing a value of zero in the INDEX_NUMBER parameter means the current index of the newly created file will also be undefined.

The index numbers supplied in the key description file are used by the NDM only to determine the boundaries between the different keys. This means that if the indices are not numbered consecutively, they will be implicitly renumbered by the NDM.

For example, if the key description file (which is usually created from the data description file created by the NDM Utilities) specifies index numbers 1, 1, 3, 3, 3, 5, 0 for its segments, the file created will have a two-segment index as index number 1, a three-segment index as index number 2, and a single-segment index as index number 3. This may cause confusion when running the NDM Utilities if the expected index numbers were 1, 3, 5.

Pre-Requisites:

- You must open the catalog file.

Currency Changes:

The current record and current index of the catalog file are changed. The current index of the newly created file is set to the value passed in INDEX_NUMBER (unless it's zero). The current record of the newly created file is "undefined".

Toolbox Features:

If the "key types" feature is enabled (feature # 2) then the indices described in the key description file parameter may contain native ISAM specific key types.

If the "file limits" feature is enabled (feature # 3) then the NDM limits are replaced by the native ISAM's limits (e.g., depending on the native ISAM, a key longer than 120 bytes may be defined).

Possible Errors:

- 10 Invalid file name.
- 37 File already exists.
- 40 Lost position. The current record of either the data description file or the key description file has been deleted, or its key has been modified.
- 42 A required record in either the key description file or the data description file is in use.
- 43 Either the key description file or the data description file is in use.
- 48 Permission error. Network privileges disallow file creation.
- 115 Index number is out of range.

See Also:

- 31020 NDM_CLOSE_FILE
- 31060 NDM_CREATE_FILE
- 31090 NDM_CREATE_KEY_TABLE
- 31150 NDM_GET_CATALOG_ENTRY
- 31290 NDM_GET_RECORD_LENGTH_FROM_DD
- 31340 NDM_OPEN_FILE

```
' 31080 NDM_CREATE_INDEX  
      (FILE_HANDLE$, KEYDESC_TABLE$, INDEX_NUMBER, RETURN_CODE)
```

Input Parameters:

FILE_HANDLE\$ - a handle to the open data file to be indexed.

KEYDESC_TABLE\$ - a key description table describing the key segments of the new index.

Output Parameters:

INDEX_NUMBER - the number of the newly created index.

Description:

This function adds a (usually) temporary index to the list of indices attached to the specified open data file. The FILE_HANDLE\$ file need not be empty to call this function. However, it must have been opened with exclusive access rights (this disallows reads as well as writes).

To create a new permanent index, we recommend that you first modify the key description file, then re-create the data file under a new name with the new index structure and copy all records from the old file to the new file.

Since this function traverses the entire data file, its operation can be quite slow for large files. However, it will usually be faster than manually sorting the file.

The segments of the key of the new index are described in KEYDESC_TABLE\$, which has the same structure as the KEYDESC_TABLE\$ parameter of **31060 NDM_CREATE_FILE**. However the newly created index's number is defined as the lowest unused index number for the FILE_HANDLE\$ file. This means that the index numbers passed in this structure are irrelevant. However they must be non-zero (except the dummy entry at the end) since a zero entry defines the end of the structure. Also, all index numbers used in this structure (except the final zero) must have the same value or an error code will be returned.

Pre-Requisites:

- The FILE_HANDLE\$ data file must be open.
- The KEYDESC_TABLE\$ variable must have been created (refer to Appendix B on the key description table).

Currency Changes:

The data file's current index is set to the newly created index. The current record is set to undefined.

Toolbox Features:

This function may only be called if the "create/delete index" toolbox feature (feature # 6) is enabled.

If the "key types" feature is enabled (feature # 2) then the indices described in the KEYDESC_TABLE\$ parameter may contain native ISAM specific key types.

If the "file limits" feature is enabled (feature # 3) then the NDM limits are replaced by the native ISAM's limits (e.g., depending on the native ISAM, a key longer than 120 bytes may be defined).

Possible Errors:

None

See Also:

31090 NDM_CREATE_KEY_TABLE
31110 NDM_DELETE_INDEX

```
' 31090 NDM_CREATE_KEY_TABLE  
      (DD_HANDLE$, KEYDESC_HANDLE$, KEYDESC_TABLE$, RETURN_CODE)
```

Input Parameters:

DD_HANDLE\$ - a handle to an open data description file that describes the fields referred to in the KEYDESC_HANDLE\$ file.

KEYDESC_HANDLE\$ - a handle to an open key description file containing the desired key information.

Output Parameters:

KEYDESC_TABLE\$ - a structure that describes the keys of a file with key description file KEYDESC_HANDLE\$. The required length of this parameter can be determined by calling **31250 NDM_GET_KEY_TABLE_SIZE**.

Description:

This function returns a key description table created from the contents of the given key description file. This table can then be used by **31060 NDM_CREATE_FILE** to describe the key structure of the file to be created.

The information read from the key description file is not checked for validity; it is merely copied into the table. Validity checking is performed by **31060 NDM_CREATE_FILE**.

Pre-Requisites:

- The data description file and key description file must be open.

Currency Changes:

The current record and current index of both the data description and key description files can be changed by this call.

Toolbox Features:

None

Possible Errors:

- 40 Lost position. The current record of either the data description or key description file was deleted, or its key was modified.
- 42 A required record in the key description or data description files is in use.

See Also:

31060 NDM_CREATE_FILE
31220 NDM_GET_FILE_STATUS
31250 NDM_GET_KEY_TABLE_SIZE

' 31100 NDM_DELETE_FILE (FILE_NAME\$, RETURN_CODE)

Input Parameters:

FILE_NAME\$ - the name of the data file to be deleted. May not include an extension (unless the "filename extension" toolbox feature is enabled).

Output Parameters:

None

Description:

This function deletes all native OS files relating to this file from the disk. The file must not be open. The name in FILE_NAME\$ should be the same name as would be passed to a call to **31340 NDM_OPEN_FILE** (i.e., it should not contain an extension, unless the "filename extensions" toolbox feature is enabled). This function will not delete the data description or key description file associated with the FILE_NAME\$ file. These files must be deleted by separate calls to DELETE_FILE.

This function should be called instead of the shell's "delete file" statement since in some native ISAMs the native OS filename might be different from the name passed to the **31340 NDM_OPEN_FILE** function, or since the native ISAM may store the file's information in more than one physical file.

Pre-Requisites:

None

Currency Changes:

None

Toolbox Features:

- If the "filename extensions" feature is enabled (feature # 1), extensions may be specified in the FILENAME\$ parameter. To specify a name with no extension, add a period at the end of the name.

Possible Errors:

- 10 Invalid file name.
- 11 The file was not found.
- 43 The file is in use.
- 110 File deletion failed. Another user may still have the file open.

See Also:

31340 NDM_OPEN_FILE



```
' 31110 NDM_DELETE_INDEX  
      ( FILE_HANDLE$, INDEX_NUMBER, RETURN_CODE )
```

Input Parameters:

FILE_HANDLE\$ - a handle to the open data file with the index to be deleted.

INDEX_NUMBER - the number of the index to be deleted.

Output Parameters:

None

Description:

This function removes the specified index from the specified data file. This function may be called on any file (empty or not) that has been opened in exclusive mode. If the current index is the one being deleted, then the file's current index and current record number become undefined.

This function should only be called to delete the last index in the list. Since it renumbers all indices that were originally numbered higher than the deleted index, each of these indices has its number lowered by one to keep the numbers contiguous. This can cause unexpected behavior if the application program expects index numbers to remain constant. We also recommend that files be kept open in exclusive mode (disallow all other reads and writes) while temporary indices for the file exist.

Pre-Requisites:

None

Currency Changes:

The current index and current record number of the data file are not changed unless the index that was deleted was the current one. Here, both the current index and current record number are set to "undefined".

Toolbox Features:

The "create/delete index" toolbox feature (feature # 6) must have been enabled for this call to work.

Possible Errors:

None

See Also:

31080 NDM_CREATE_INDEX

' 31120 NDM_DELETE_RECORD (FILE_HANDLE\$, RETURN_CODE)

Input Parameters:

FILE_HANDLE\$ - the handle of the open file containing the record to be deleted.

Output Parameters:

None

Description:

This function deletes the current record of the specified file.

Pre-Requisites:

- The FILE_HANDLE\$ file must be open.
- The current record of the file must be the record to be deleted.
- The current index of the file must be set to the desired value.

Currency Changes:

The file's current record is unchanged by this call. A subsequent call to **31370 NDM_READ_BY_POSITION** which attempts to read the current record will result in an error. However, "previous" and "next" reads will function properly.

Toolbox Features:

None

Possible Errors:

42 The record is in use.

See Also:

None

```
' 31130 NDM_DESTROY_CONVERSION_TABLE  
      ( CONVERSION_TABLE$, RETURN_CODE )
```

Input Parameters:

CONVERSION_TABLE\$ - the handle to the conversion table to be destroyed.

Output Parameters:

None

Description:

This function frees up the space used by the specified conversion table handle. Future access to this conversion table handle will result in an error.

Typically this function should be called for all conversion tables associated with a file that is being closed.

Pre-Requisites:

- The CONVERSION_TABLE\$ handle must have been created by a call to either **31050 NDM_CREATE_CONVERSION_TABLE** or **31040 NDM_CREATE_CONV_TABLE_FOR_KEY**.

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

None

See Also:

31030 NDM_CREATE_CONVERSION_TABLE
31170 NDM_GET_CONVERSION_TABLE_LIST

' 31140 NDM_GET_ISAM_ERROR_CODE (ERROR_CODE)

Input Parameters:

None

Output Parameters:

ERROR_CODE - the native ISAM error code (if any) generated by the last NDM call performed.

Description:

This function returns the native ISAM error code corresponding to the native ISAM error (if any) encountered during the last call to any API routine.

NOTE: This function will usually only be called when an API function call has indicated, via its RETURN_CODE parameter, that a low-level (native ISAM) error has occurred. Also, the same error condition will produce different codes as an API error ('RETURN_CODE') or as a native ISAM 'ERROR_CODE', since the API error code is native ISAM independent.

Typically this function will be used to provide details to the operator regarding an unexpected error condition.

Pre-Requisites:

None

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

None

See Also:

31210 NDM_GET_ERROR_DESCRIPTION

```
' 31150 NDM_GET_CATALOG_ENTRY
      (CATALOG_HANDLE$, FILE_TITLE$, FILE_NAME$, DD_NAME$,
       KEYDESC_NAME$, ISAM_SPECIFIC$, RETURN_CODE)
```

Input Parameters:

CATALOG_HANDLE\$ - the handle to the open catalog file to be read.

FILE_TITLE\$ - a 32-character mnemonic name for the file, which is matched against the "File Title" field of the catalog file.

Output Parameters:

FILE_NAME\$ - the name of the file, without an extension (unless the "filename extensions" toolbox feature is enabled).

DD_NAME\$ - the name of the data description file for this file.

KEYDESC_NAME\$ - the name of the key description file for this file.

ISAM_SPECIFIC\$ - a 128-byte table containing native ISAM specific information used in file creation.

Description:

This function looks up the file with the specified title in the specified catalog file and returns that file's information.

Calling this function is equivalent to (manually):

```
Setting the current index of the catalog file to 1
Calling 31360 NDM_READ_BY_KEY with the FILE_TITLE$
Separating the returned record into the above fields.
```

Pre-Requisites:

- The catalog file must be open.

Currency Changes:

The current record and current index of the catalog file are changed by this call.

Toolbox Features:

None

Possible Errors:

42 The catalog file record is in use.

See Also:

31060 NDM_CREATE_FILE

31340 NDM_OPEN_FILE

```
' 31160 NDM_GET_CONFIGURATION
      ( CONVERSION_TABLE_COUNT, CONVERSION_ENTRY_COUNT,
        FILE_COUNT, RECORD_SIZE, ISAM_NAME$, ISAM_CODE, API_REV$,
        API_SERIAL$, USER_COUNT, USER_LIMIT, NDM_DIRECTORY$,
        RETURN_CODE )
```

Input Parameters:

None

Output Parameters:

CONVERSION_TABLE_COUNT - the value passed in the CONVERSION_TABLE_COUNT parameter of the last call to **31320 NDM_INITIALIZE**.

CONVERSION_ENTRY_COUNT - the value passed in the CONVERSION_ENTRY_COUNT parameter of the last call to **31320 NDM_INITIALIZE**.

FILE_COUNT - the value passed in the FILE_COUNT parameter of the last call to **31320 NDM_INITIALIZE**.

RECORD_SIZE - the value passed in the RECORD_SIZE parameter of the last call to **31320 NDM_INITIALIZE**.

ISAM_NAME\$ - a 72-character string containing the name and release number of the native ISAM in use by the NDM.

ISAM_CODE - a numeric code identifying the native ISAM in use (irrespective of release number). As of NDM Revision 1.0, the following ISAM_CODE's may be returned:

- 1 BTRIEVE
- 2 C-ISAM
- 5 SDTRIEVE

API_REV\$ - a 72-character string indicating the revision of the API in use.

API_SERIAL\$ - a 6-character string indicating the serial number of the API in use.

USER_COUNT - the number of users actively using NDM when NDM was loaded.

USER_LIMIT - the maximum number of NDM users allowed.

NDM_DIRECTORY\$ - a 72-character blank-padded string containing the name of the directory containing the NDM files such as the NDM data dictionary system files, as set by the NDM environment variable.

Description:

This function returns the information passed to the last call to **31320 NDM_INITIALIZE**, as well as some site-specific information such as the native ISAM in use and the directory containing NDM system files.

NOTE: The use of the native ISAM information returned by this function, for other than display purposes, may make an application native ISAM dependent.

Pre-Requisites:

None

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

None

See Also:

31320 NDM_INITIALIZE

```
' 31170 NDM_GET_CONVERSION_TABLE_LIST  
      ( CONVERSION_TABLE_COUNT, CONVERSION_TABLE_LIST$,  
        RETURN_CODE )
```

Input Parameters:

None

Output Parameters:

CONVERSION_TABLE_COUNT - the number of conversion tables currently in use.

CONVERSION_TABLE_LIST\$ - a concatenated list of two-byte handles to all conversion tables currently in use.

Description:

This function returns a sequential list of handles to all conversion tables currently in use. This parameter should be dimensioned large enough to be able to hold as many handles as was specified in the CONVERSION_TABLE_COUNT parameter of the last call to **31320 NDM_INITIALIZE**. CONVERSION_TABLE_COUNT receives the number of conversion table handles in CONVERSION_TABLE_LIST\$.

This function will not be used by most applications. Its main use is for applications that require detailed control over allocation and de-allocation of conversion tables, usually due to memory constraints. This function can also be useful for debugging purposes to check that the NDM list of conversion tables matches the application's list.

Pre-Requisites:

- Must be preceded by one or more calls to **31050 NDM_CREATE_CONVERSION_TABLE** or **31040 NDM_CREATE_CONV_TABLE_FOR_KEY**.

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

None

See Also:

31040 NDM_CREATE_CONV_TABLE_FOR_KEY
31050 NDM_CREATE_CONVERSION_TABLE
31130 NDM_DESTROY_CONVERSION_TABLE
31320 NDM_INITIALIZE

' 31180 NDM_GET_CURRENT_INDEX (FILE_HANDLE\$, INDEX_NUMBER, RETURN_CODE)

Input Parameters:

FILE_HANDLE\$ - the handle of the open file to be queried.

Output Parameters:

INDEX_NUMBER - the number of the index currently in use for this file.

Description:

This function returns the number of the current index for the specified file. If the current index is invalid, a zero is returned.

Pre-Requisites:

- The FILE_HANDLE\$ file must be open.

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

None

See Also:

31400 NDM_SET_CURRENT_INDEX

```
' 31200 NDM_GET_DEFAULT_FIELD_TYPE  
      (FIELD_CLASS$, FIELD_TYPE, RETURN_CODE)
```

Input Parameters:

FIELD_CLASS\$ - contains one of the following:

- A return the default native alpha field type.
- N return the default native numeric field type.
- D return the default native data type.

Output Parameters:

FIELD_TYPE - a numeric "field type" code, useable in a data description.

Description:

This function returns a default native field type based on the first encountered record in the "Field Types" file that has a "Y" in its DEFAULT TYPE field and has the specified FIELD_CLASS. This is provided as a convenience for those who don't want to select their own native field types. The value returned by this function for FIELD_CLASS\$ = "N" will be a general-purpose numeric type, suitable for use by commercial applications. Wherever possible it will be identical with a currently supported Basic-2C numeric field type such as signed packed decimal (5dxx).

If the "Field Type" file cannot be accessed, a built-in native ISAM dependent default value will be returned.

Pre-Requisites:

None

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

- 40 Lost position. The current record of the field types file either was deleted, or its key was modified.

- 42 The record of the field types file is in use.

See Also:

None

```
' 31210 NDM_GET_ERROR_DESCRIPTION  
      (ERROR_TYPE, ERROR_CODE, ERROR_DESCRIPTION$, RETURN_CODE)
```

Input Parameters:

ERROR_TYPE - contains one of the following:

- 0 ERROR_CODE is an API error (as returned in a RETURN_CODE variable).
- 1 ERROR_CODE is a native ISAM error (as returned by **31140 NDM_GET_ISAM_ERROR_CODE**).

ERROR_CODE - the error code to look up.

Output Parameters:

ERROR_DESCRIPTION\$ - a 72-character textual description of the specified error.

Description:

This function returns a textual description of the specified native ISAM or API error, retrieved from the "Error Description" NDM support file.

NOTE: An error type value of **1** is used to access the native ISAM error description regardless of which native ISAM is in use.

Pre-Requisites:

None

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

- 40 Lost position. The current record of the error description file was deleted or its key was changed.
- 42 The record of the error description file is in use.
- 43 The error description file is in use.

See Also:

31140 NDM_GET_AM_ERROR_CODE


```
' 31220 NDM_GET_FILE_STATUS
      (FILE_HANDLE$, STATUS_TABLE$, ISAM_SPECIFIC$,
       RECORD_COUNT, RECORD_LENGTH, FILE_NAME$, MODE$, EOF$,
       BOF$, RETURN_CODE)
```

Input Parameters:

FILE_HANDLE\$ - the handle to the open file to be queried.

Output Parameters:

STATUS_TABLE\$ - a 400 byte key description table describing the indices currently in use by the native ISAM.

ISAM_SPECIFIC\$ - a 128-byte table containing the information passed in the ISAM_SPECIFIC\$ parameter when this file was created.

RECORD_COUNT - the number of data records in the file.

RECORD_LENGTH - the number of bytes in a data record in this file.

FILE_NAME\$ - the name of the file as passed to **31340 NDM_OPEN_FILE** when the file was opened.

MODE\$ - single character containing the mode (S, X, or R) that was passed to **31340 NDM_OPEN_FILE**.

EOF\$ - single character containing "Y" if the file is at EOF and "N" otherwise.

BOF\$ - single character containing "Y" if the file is at BOF and "N" otherwise.

Description:

This call returns the information that was specified when a file was created and/or indexed.

NOTE: The STATUS_TABLE\$ returned by this call describes all active indices, which may differ from the information in the data file's key description file if the key description file has been edited or if indices have been created or destroyed.



Do not use the value returned by 31250 NDM_GET_KEY_TABLE_SIZE to dimension the STATUS_TABLE\$ in this call! 31250 NDM_GET_KEY_TABLE_SIZE returns the size of the key description stored in the key description file, while 31220 NDM_GET_FILE_STATUS returns the key information actually used in the file. The STATUS_TABLE\$ should always be dimensioned to 400 bytes (24 entries plus one end marker, times 16 bytes per entry).

Pre-Requisites:

- The FILE_HANDLE\$ file must be open.
- The STATUS_TABLE\$ must be dimensioned to 400 bytes.

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

None

See Also:

31060 NDM_CREATE_FILE

```
' 31223 NDM_GET_FORMAT_SPEC_FOR_KEY
      ( DD_HANDLE$, KEYDESC_HANDLE$,
        INDEX_NUMBER, FORMAT_SPEC$, KEY_IN_RECORD$, MAX_ALPHA_LEN,
        FIELD_COUNT, FIELD_OK$, RETURN_CODE )
```

Input Parameters:

DD_HANDLE\$ - the handle for the open data description file containing conversion information for the desired fields.

KEYDESC_HANDLE\$ - the handle to the open key description file containing a description of the key to create a format specification for.

INDEX_NUMBER - the number to match the "INDEX NUMBER" field of the key description record.

KEY_IN_RECORD\$ - contains "Y" if the fields of the key will be in their correct places in the Basic-2C record, or "N" if they will be contiguous at the start of the record.

MAX_ALPH_LEN - the maximum length to use for an ALPHA (A0xx) \$PACK specification. Fields that are longer than this length will be split into two or more ALPHA specifications with the total of all of the lengths adding up to the length of the field.

Output Parameters:

FORMAT_SPEC\$ - the \$PACK format specification containing one \$PACK "field form" parameter for each field in the key. This string can be used as the "pack specification" by \$PACK and/or \$UNPACK.

FIELD_COUNT - the number of "field form" parameter specifications in FORMAT_SPEC\$.

FIELD_OK\$ - "Y" if all of the field types of the key segments could be translated into valid \$PACK codes, "N" if not. For fields that cannot be represented by a \$PACK "form field", a skip code (00xx) is put in.

Description:

This function creates a \$PACK format specification for the key segments of a particular index. This format specification can then be used to access the individual fields in a key.

The specified key description file is searched for entries with the specified INDEX_NUMBER. For each of these, the entry in the data description file with a FIELD_NAME matching the key description file's entry is used to create a \$PACK "field form" specification.

If the keys to be used will be placed into their normal positions in a Basic-2C record, then a "Y" should be passed in KEY_IN_RECORD\$. If the key segments are not contiguous within the record then \$PACK skip (00xx) codes will be put into the FORMAT_SPEC\$ between the key segment codes wherever necessary. If a "N" is passed in this in KEY_IN_RECORD\$, it is assumed that key segments will be placed contiguously into the Basic-2C record, in order of segment number and therefore no skip codes will be generated.

The FORMAT_SPEC\$ can be used in the \$PACK and \$UNPACK statements to access the key segment fields from a record that has been converted to Basic-2C format.

Pre-Requisites:

- The data description and key description files must be open.

Currency Changes:

The current index and current record of the data description and key description files are changed.

Toolbox Features:

None

Possible Errors:

- 40 Lost position. The current record of the data description or key description file was deleted, or its key was modified.
- 42 A required data description or key description record is in use.
- 43 Either the data description file or the key description file is in use.

See Also:

- 31030 NDM_CONVERT
- 31223 NDM_GET_FORMAT_SPEC

```
' 31225 NDM_GET_FORMAT_SPEC
      (DD_HANDLE$, FIELD_NAMES$, CONVERSION_TABLE_NUMBER,
       FORMAT_SPEC$, MAX_ALPHA_LEN, FIELD_COUNT, FIELD_OK$,
       RETURN_CODE)
```

Input Parameters:

DD_HANDLE\$ - the handle for the open data description file describing the data record to create a format specification for.

FIELD_NAMES\$ - a list of field names, as stored in the data description, to be used in the format specification. Multiple names are separated by commas. If this parameter is blank, all fields with the specified conversion table number will be used.

CONVERSION_TABLE_NUMBER - the number of the conversion table to create a format specification for. This value is matched against the "conversion table number" field of the data description records. If names are specified in FIELD_NAMES\$, this value is ignored.

MAX_ALPHA_LEN - the maximum length to use for an ALPHA (A0xx) \$PACK specification. Fields that are longer than this length will be split into two or more ALPHA specifications with the total of all of the lengths adding up to the length of the field.

Output Parameters:

FORMAT_SPEC\$ - the \$PACK format specification containing one \$PACK "field form" parameter for each data description field. This string can be used as the "pack specification" by \$PACK and/or \$UNPACK.

FIELD_COUNT - the number of "field form" parameter specifications in FORMAT_SPEC\$.

FIELD_OK\$ - "Y" if all of the data description field types could be translated into valid \$PACK codes, "N" if not. For fields that cannot be represented by a \$PACK "form field", a skip code (00xx) is put in.

Description:

This function reads the conversion information in the specified data description file and creates a \$PACK format specification that can be used to extract the individual fields from a data record. Format specifications are only created for records in the data description file whose "conversion table number" field matches the CONVERSION_TABLE_NUMBER parameter, or whose names are specified in FIELD_NAMES\$ (if any). The FORMAT_SPEC\$ can be used in the \$PACK and \$UNPACK statements to access the fields from a record that has been converted to Basic-2C format.

Pre-Requisites:

- The data description file must be open.

Currency Changes:

The current index and current record of the data description file are changed.

Toolbox Features:

None

Possible Errors:

- 40 Lost position. The current record of the data description file was deleted, or its key was modified.
- 42 A required data description record is in use.
- 43 The data description file is in use.

See Also:

- 31030 NDM_CONVERT
- 31223 NDM_GET_FORMAT_SPEC_FOR_KEY

```
' 31230 NDM_GET_HANDLE_SIZE  
      (HANDLE_SIZE, RETURN_CODE)
```

Input Parameters:

None

Output Parameters:

HANDLE_SIZE - the number of bytes required to store a file handle.

Description:

This function returns the minimum size of a variable required for use as a FILE_HANDLE\$. This value can then be used to DIM a string for use as a FILE_HANDLE\$ in a subsequent **31340 NDM_OPEN_FILE** call. This routine will normally only be used if the NDM system has been initialized (via a call to **31320 NDM_INITIALIZE**) with zero specified as the maximum number of open files. In this case, a file handle will contain a native ISAM file handle to the data file, plus whatever other information is required by the API. If a non-zero number of files required is specified to **31320 NDM_INITIALIZE**, then the file handle merely contains a pointer to an internal NDM data area and its size (two bytes) is therefore native ISAM independent.

Pre-Requisites:

None

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

None

See Also:

31320 NDM_INITIALIZE

31340 NDM_OPEN_FILE

' 31240 NDM_GET_INDEX_NUMBER (KEYDESC_HANDLE\$, KEYNAME\$, INDEX_NUMBER, RETURN_CODE)
--

Input Parameters:

KEYDESC_HANDLE\$ - the handle to the open key description file to be searched.

KEYNAME\$ - a 32-character string, to be matched against the "FIELD NAME" field of the key description file record.

Output Parameters:

INDEX_NUMBER - the number of the index with the given name, or -1 if there is no index with that name.

Description:

This function searches a key description file for an index with a given name, and returns the number of that index if found. The key description file must be open. It is searched for a record with a "segment number" field of 0 and a "field name" field matching the value in KEYNAME\$.

This routine may change the "current record" pointer and/or the current index number of the KEYDESC_HANDLE\$ file.

Pre-Requisites:

- The key description file must be open.

Currency Changes:

The current record and current index of the key description file may be changed.

Toolbox Features:

None

Possible Errors:

None

See Also:

None


```
' 31250 NDM_GET_KEY_TABLE_SIZE  
      (KEYDESC_HANDLE$, KEY_TABLE_SIZE, RETURN_CODE)
```

Input Parameters:

KEYDESC_HANDLE\$ - a handle to an open key description file.

Output Parameters:

KEY_TABLE_SIZE - the size required by a key table for the KEYDESC_HANDLE\$ file.

Description:

This function returns the number of bytes needed for the key description table in a call to **31090 NDM_CREATE_KEY_TABLE**. This value can be used to DIM a string to be passed as the KEY_TABLE\$ parameter to **CREATE_KEY_TABLE**.

If desired, the application programmer can simply declare key description tables to be the maximum size allowable (currently 400 bytes) instead of calling this function for each key description table.

Pre-Requisites:

- The key description file must be open.

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

None

See Also:

31090 NDM_CREATE_KEY_TABLE

```
' 31260 NDM_GET_LIMIT_HIGHWATERS  
      (HIGHWATER_TABLE$, RETURN_CODE)
```

Input Parameters:

None

Output Parameters:

HIGHWATER_TABLE\$ - a "limits table" containing the highest used value of each limit since the last call to **31320 NDM_INITIALIZE**.

Description:

This function returns a "limits" structure giving the maximum used value of certain limits imposed by the API (e.g., maximum keys per file). This function will usually be called after enabling the "file limits" toolbox feature (although this is not required), since the limits that routine extends are the same limits that go into HIGHWATER_TABLE\$. This function is typically used to provide information about the portability of an application. Refer to Appendix B for a detailed description of this table (Limits Table).

Pre-Requisites:

None

Currency Changes:

None

Toolbox Features:

This function is typically called when the "file limits" toolbox feature (feature # 3) is enabled. However, it will work even if this feature is disabled.

Possible Errors:

None

See Also:

31310 NDM_GET_TOOLBOX_STATUS
31420 NDM_SET_TOOLBOX_STATUS

' 31270 NDM_GET_OPEN_FILE_LIST (OPEN_FILE_COUNT, OPEN_FILE_LIST\$, RETURN_CODE)
--

Input Parameters:

None

Output Parameters:

OPEN_FILE_COUNT - the number of files that are currently open by this application.

OPEN_FILE_LIST\$ - a list of concatenated two-byte handles to all currently open files.

Description:

This function returns a sequential list of file handles corresponding to the files that are currently open. This parameter should be dimensioned large enough to be able to hold as many handles as was specified in the FILE_COUNT parameter to **31320 NDM_INITIALIZE**. If this value were zero, then zero will be passed back to OPEN_FILE_COUNT and OPEN_FILE_LIST\$ will be unchanged.

This function will not be used by most applications.

This function is useful for debugging purposes to check that the NDM list of file handles matches the application's list.

Pre-Requisites:

None

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

None

See Also:

31020 NDM_CLOSE_FILE
31230 NDM_GET_HANDLE_SIZE
31320 NDM_INITIALIZE
31340 NDM_OPEN_FILE

' 31280 NDM_GET_POSITION (FILE_HANDLE\$, POSITION_HANDLE\$, RETURN_CODE)

Input Parameters:

FILE_HANDLE\$ - the handle of the open file whose current position is desired.

Output Parameters:

POSITION_HANDLE\$ - an eight-byte value indicating the current position in the file.

Description:

This function returns a pointer into the specified file that indicates the current position in that file. This value is an eight-byte string.

NOTE: The only valid operation on this value is comparison for equality - "less than" or "greater than" comparisons will not necessarily reflect the relative positions of records in the file.

The returned value can also be used when calling **31410 NDM_SET_POSITION**.

This function is typically used when an application needs to "remember" the location of a particular record and either there is no unique key for the file or the application cannot "remember" the key.

This function should only be used for temporary storage of a record's location. To ensure that the record has not been modified by other users, either the record should be locked by the application for the duration of the "get position/set position" operation or the application must reread the record after the "set position" call.

Pre-Requisites:

- The FILE_HANDLE\$ file must be open.

Currency Changes:

None

Toolbox Features:

The "get/set position" toolbox feature (feature # 5) must be enabled to use this function.

Possible Errors:

None

See Also:

31410 NDM_SET_POSITION

```
' 31290 NDM_GET_RECORD_LENGTH_FROM_DD  
      (DD_HANDLE$, RECORD_LENGTH, RETURN_CODE)
```

Input Parameters:

DD_HANDLE\$ - the handle to the open data description file to be read.

Output Parameters:

RECORD_LENGTH - the length of the record as defined in the specified data description file.

Description:

This function returns the record length of the specified file, determined by finding the record whose "conversion table number" field is zero and has the highest start position - its start position and length are added and the sum is decreased by one. This result is returned in RECORD_LENGTH.

Pre-Requisites:

- The data description file must be open.

Currency Changes:

The current index and current record of the data description file may be changed.

Toolbox Features:

None

Possible Errors:

- 40 Lost position. The current record of the data description file was deleted, or its key was changed.
- 42 The required record in the data description file is in use.

See Also:

31150 NDM_GET_CATALOG_ENTRY
31220 NDM_GET_FILE_STATUS

' 31310 NDM_GET_TOOLBOX_STATUS (FEATURE_NUMBER, TOOLBOX_STATUS, RETURN_CODE)

Input Parameters:

FEATURE_NUMBER: a code identifying the toolbox feature to be looked up. For a description of each feature refer to Chapter 5 on the toolbox feature.

Possible values include:

- 1 Filename extensions
- 2 Key types
- 3 File limits
- 4 Transactions
- 5 Get/Set position
- 6 Create/Delete index

Output Parameters:

TOOLBOX_STATUS - contains one of the following:

- 1 the toolbox routines are available.
- 0 the toolbox routines are unavailable.

Description:

This function returns a value reflecting the current accessibility of a toolbox feature. The default status of each feature, set by a call to **31320 NDM_INITIALIZE**, is zero (unavailable).

Pre-Requisites:

None

Currency Changes:

None

Toolbox Features:

This function allows the application to inspect the accessibility of each toolbox feature.

Possible Errors:

None

See Also:

31320 NDM_INITIALIZE
31420 NDM_SET_TOOLBOX_STATUS

```
' 31315 NDM_GET_TRANSLATION_TABLE  
      (NAME$, B2C_TO_NDM$, NDM_TO_B2C$, RETURN_CODE)
```

Input Parameters:

None

Output Parameters:

NAME\$ - the name of the translation table that is currently being used (32 bytes)

B2C_TO_NDM\$ - the translation table for converting from Basic-2C to native (256 bytes)

NDM_TO_B2C\$ - the translation table for converting from native to Basic-2C (256 bytes)

Description:

This function gets the name and contents of the translation table currently being used.

Pre-Requisites:

None

Currency Changes:

None

Toolbox features:

None

Possible Errors:

None

See Also:

31425 NDM_SET_TRANSLATION_TABLE

```
' 31320 NDM_INITIALIZE  
      ( CONVERSION_TABLE_COUNT, CONVERSION_ENTRY_COUNT,  
        FILE_COUNT, RECORD_SIZE, RETURN_CODE )
```

Input Parameters:

CONVERSION_TABLE_COUNT - the maximum number of conversion tables that will be in use at once during this application.

CONVERSION_ENTRY_COUNT - the maximum number of conversion table entries (data fields) that will be in use in all conversion tables.

FILE_COUNT - the maximum number of files that will be open concurrently by the application. This includes data dictionary files such as data description and key description files, but not data dictionary system files such as the key description data description file (unless these are opened explicitly by the application). If this value is zero, file handles are stored in Basic-2C space and thus their number is limited only by available memory.

RECORD_SIZE - the number of bytes in the longest record (either native or Basic-2C) used by this application. A buffer of this size will be allocated in NDM memory.

Output Parameters:

None

Description:

This routine must be called before any other NDM routine is called. This provides NDM with information required to set up its internal data structures. It reads the contents of the NDM data dictionary system files and stores them in an internal buffer.

If the FILE_COUNT parameter is set to zero, then all of the file-related information will be stored in the file handle (in Basic-2C space), minimizing the memory required by the API and thus giving the application programmer more control over memory usage. Here, the only limit on the number of open files is the amount of memory available to the application.

Calling this function also turns off all toolbox features.

Calls to this function after the initial call will close the NDM system in use before starting a new one. That is, all files, locks, and conversion tables will be closed, and all

internal data will be reset. If called from inside a transaction, this function will close the transaction by calling **31440 NDM_TRANSACTION_COMPLETE**.

Pre-Requisites:

None

Currency Changes:

None

Toolbox Features:

This function disables all toolbox features.

Possible Errors:

None

See Also:

31160 NDM_GET_CONFIGURATION
31310 NDM_GET_TOOLBOX_STATUS
31420 NDM_SET_TOOLBOX_STATUS

```
' 31330 NDM_INSERT_RECORD  
      (FILE_HANDLE$, RECORD_BUFFER$, LOCK$, RETURN_CODE)
```

Input Parameters:

FILE_HANDLE\$ - the handle to the open file to receive the new record.

RECORD_BUFFER\$ - the new record to be inserted, in native ISAM format.

LOCK\$ - contains one of the following values:

- L lock this record if it was written successfully.
- " " don't lock this record.

Output Parameters:

None

Description:

This function inserts the record data in RECORD_BUFFER\$ into the specified open file. The current record will be set to the newly inserted record upon return. This function does not perform any conversion - if you want to insert a converted record you must call **31030 NDM_CONVERT** before this call. The LOCK\$ parameter is a single character - if it contains an "L" then the newly inserted record will be locked.

An optional method of adding a record to a file, intended primarily for applications that use DC-style Basic-2C data management, is **31010 NDM_APPEND_UNIQUE_KEY_RECORD**.

Locked records may be unlocked by calling either **31380 NDM_REWRITE_RECORD** or **31460 NDM_UNLOCK_ALL_RECORDS**.

Pre-Requisites:

- The FILE_HANDLE\$ file must be open.
- The RECORD_BUFFER\$ variable must contain native data (usually created by calling **31030 NDM_CONVERT**).

Currency Changes:

The data file's current record is set to the newly inserted record.

Toolbox Features:

None

Possible Errors:

7 Duplicate keys are not allowed.

See Also:

31010 NDM_APPEND_UNIQUE_KEY_RECORD

31030 NDM_CONVERT

31380TNDM_REWRITE_RECORD

```
' 31340 NDM_OPEN_FILE  
      ( FILE_NAME$, MODE$, INDEX_NUMBER, FILE_HANDLE$,  
        RETURN_CODE )
```

Input Parameters:

FILE_NAME\$ - the name of the file to be opened. This does not include an extension (unless the "filename extensions" toolbox feature is enabled).

MODE\$ - a one character string giving the type of access desired.

- X exclusive access. No others may open the file (in any mode) while this task has it open.
- S shared mode. Any other tasks not requesting exclusive mode may open this file concurrently.
- R read only mode. May be used in cases where the user does not have write access to the file.

INDEX_NUMBER - the value to set the "current index" of the opened file to.

Output Parameters:

FILE_HANDLE\$ - a handle identifying the file that was opened by this call. The size required of a handle may be determined by calling **31230 NDM_GET_HANDLE_SIZE**.

Description:

This function opens the data file with the specified native OS filename and sets the current index to the given value.

A file lock, specified if MODE\$ = "X", is a "no wait" lock - if the lock request fails, this routine will return immediately with an appropriate code in RETURN_CODE.

The current record of the file is undefined when the file is first opened.

If the specified index number is zero or does not correspond to a valid index (e.g., if it contains extended key types and the "data types" toolbox feature is disabled), the file is opened with its current index set to "undefined".

Pre-Requisites:

- Optional. The size required of a file handle can be found by calling **31230 NDM_GET_HANDLE_SIZE**.

Currency Changes:

The current index of the newly opened file is set to the value passed in INDEX_NUMBER if this corresponds to a valid index (it might not if the "key types" toolbox feature is disabled). The current record is set to "undefined".

Toolbox Features:

If the "filename extensions" feature is enabled (feature # 1), extensions may be specified in the FILENAME\$ parameter. To specify a name with no extension, and to avoid the addition of a default extension, add a period at the end of the name.

If the "key types" feature is enabled (feature # 2) then the INDEX_NUMBER index may contain native ISAM specific key types.

If the "file limits" feature is enabled (feature # 3) then the NDM limits are replaced by the native ISAM's limits (e.g., depending on the native ISAM, a key longer than 120 bytes may exist in the file.)

Possible Errors:

- 10 Invalid file name.
- 11 The file was not found.
- 24 The file was not created by this native ISAM.
- 43 The file is in use.
- 48 Permission error. Network privileges disallow file access.
- 122 Invalid file type. Unsupported native ISAM specific features are in use.

See Also:

31230 NDM_GET_HANDLE_SIZE


```
' 31360 NDM_READ_RECORD_BY_KEY
      ( FILE_HANDLE$, RECORD_BUFFER$, SEARCH_TYPE$, LOCK$,
        RETURN_CODE )
```

Input Parameters:

FILE_HANDLE\$ - the handle to the open file to read from.

RECORD_BUFFER\$ - a "dummy" data record containing only the key value to be searched for, in its normal position in the record.

SEARCH_TYPE\$ - contains one of the following:

- = the record's key must match SEARCH_RECORD\$ exactly.
- > the record's key must follow the key in SEARCH_RECORD\$.
- < the record's key must precede the key in SEARCH_RECORDS.
- > = the record's key must be equal to or follow the key in SEARCH_RECORDS.
- < = the record's key must be equal to or precede the key in SEARCH_RECORDS.

LOCK\$ - a one-character string containing one of:

- L lock this record if it was read successfully.
- " " don't lock this record.

Output Parameters:

RECORD_BUFFER\$ - contains the record that was read in, in native ISAM format.

Description:

This function finds and reads the first record in the specified file that matches the given criteria. The key of each record is compared to the key value(s) in RECORD_BUFFER\$. RECORD_BUFFER\$ is a dummy record that contains only the key being searched for. All segments of the key must be embedded in the proper location within the record.

If descending order was specified for a key, then calling this routine with SEARCH_TYPE\$ of "> " will produce a record whose key is **less than** the value in KEY_VALUE\$. In general, "> " and "< " refer to "following" and "preceding", respectively, in the collating sequence of the index.

The value you pass in RECORD_BUFFER\$ is in native ISAM format. This means that if your keys are in Basic-2C format you must call **31030 NDM_CONVERT** before passing the record to this function.

Record locks are "no wait" locks - if the lock fails, this routine will immediately return with an appropriate code in RETURN_CODE. One record per file may be locked. A locked record can be unlocked by calling **31460 NDM_UNLOCK_ALL_RECORDS**.

Locked records may be unlocked by calling:

31360 NDM_READ_RECORD_BY_KEY
31370 NDM_READ_RECORD_BY_POSITIONR31380 NDM_REWRITE_RECORD
31460 NDM_UNLOCK_ALL_RECORDS

Pre-Requisites:

- The FILE_HANDLE\$ file must be open.
- The current index of this file must be set to the desired value.
- Optional - The key value in SEARCH_RECORD\$ can be converted from Basic-2C format to native format by calling **31030 NDM_CONVERT**.

Currency Changes:

The current record is set to the first matching record found (if any). If no match is found, the current record is set to EOF (for forward searches), BOF (for backward searches), or undefined (for exact matches).

Toolbox Features:

None

Possible Errors:

- 1 End of file was encountered.
- 2 Beginning of file was encountered.
- 3 There are no records in the file.
- 6 A record with the desired key value was not found.
- 40 Lost position. The current record was deleted, or its key was modified.
- 42 The record is in use.

See Also:

31030 NDM_CONVERT

```
' 31370 NDM_READ_RECORD_BY_POSITION
      ( FILE_HANDLE$, RECORD_BUFFER$, POSITION, LOCK$,
        RETURN_CODE )
```

Input Parameters:

FILE_HANDLE\$ - the handle of the open file to be read from.

POSITION - specifies how to move the current record pointer before reading. Contains one of the following:

- 0 don't change the current record.
- 1 move forward one record along the current index.
- 1 move backward one record along the current index.
- 2 move to the last record of the file.
- 2 move to the first record of the file.

LOCK\$ - a one-character string containing one of:

- L lock this record if it was read successfully.
- " " don't lock this record.

Output Parameters:

RECORD_BUFFER\$ - the data record that was read in, in native ISAM format.

Description:

This function changes the current record of the given file and then reads a record from it into RECORD_BUFFER\$.

Record locks are "no wait" locks - if the lock fails, this routine will immediately return with an appropriate code in RETURN_CODE. One record per file may be locked. A locked record can be unlocked by calling **31460 NDM_UNLOCK_ALL_RECORDS**.

Locked records may be unlocked by calling:

```
31360 NDM_READ_RECORD_BY_KEY
31370 NDM_READ_RECORD_BY_POSITION 31380 NDM_REWRITE_RECORD
31460 NDM_UNLOCK_ALL_RECORDS
```

Pre-Requisites:

- The FILE_HANDLE\$ file must be open.

- For POSITION values 0, 1, or -1 only, the current record must be valid.
- The current index of this file must be set to the desired value.

Currency Changes:

The current record is changed as described.

Toolbox Features:

None

Possible Errors:

- 1 End of file.
- 2 Beginning of file.
- 3 There are no records in the file.
- 40 Lost position. The current record was deleted, or its key was modified.
- 42 The record is in use.

See Also:

- 31380 NDM_REWRITE_RECORD
- 31460 NDM_UNLOCK_RECORD

' 31380 NDM_REWRITE_RECORD (FILE_HANDLE\$, RECORD_BUFFER\$, RETURN_CODE)

Input Parameters:

FILE_HANDLE\$ - the handle to the open file to be written to.

RECORD_BUFFER\$ - the data to be written to the file, in native ISAM format.

Output Parameters:

None

Description:

This function writes the specified data onto the current record of the specified file.

The data in RECORD_BUFFER\$ must be in "native" format - this means that to write a Basic-2C record with this function you must first call **31030 NDM_CONVERT**.

To ensure program correctness, you should not call this function unless the record was locked when it was read in. This is to ensure that the record's contents have not been changed since it was read in - if the record was not locked, a call to **31380 NDM_REWRITE_RECORD** would erase the changes made by someone else.

Pre-Requisites:

- The FILE_HANDLE\$ file must be open.
- The current record of this file should be set to the record to be written.

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

- 1 End of file.
- 2 Beginning of file.
- 3 The file contains no records.
- 7 Duplicate keys are not allowed.
- 38 Record conflict. This record was changed since the last time it was read. This error could occur on native ISAMs that support passive concurrency. Not all native ISAMs support this feature.
- 40 Lost position. The current record was deleted, or its key was modified.
- 42 The current record is in use.

See Also:

- 31030 NDM_CONVERT
- 31360 NDM_READ_RECORD_BY_KEY
- 31370 NDM_READ_RECORD_BY_POSITION

```
' 31400 NDM_SET_CURRENT_INDEX  
      ( FILE_HANDLE$, INDEX_NUMBER, RETURN_CODE )
```

Input Parameters:

FILE_HANDLE\$ - the handle of the open file that is to have its current index changed.

INDEX_NUMBER - the value to set the current index of the FILE_HANDLE\$ file to.

Output Parameters:

None

Description:

This function sets the current index number of the specified file to the specified value. The file's current record pointer is set to "invalid". The current index's value can later be read by calling **31180 NDM_GET_CURRENT_INDEX**.

If INDEX_NUMBER is zero, the current index and current record are undefined.

Pre-Requisites:

- The FILE_HANDLE\$ file must be open.

Currency Changes:

The current index of the file is set to the desired value (or "invalid" if zero or an invalid index). The current record is set to "invalid".

Toolbox Features:

If the "key types" toolbox feature (feature # 2) is enabled, the INDEX_NUMBER index may contain native ISAM specific key types.

Possible Errors:

3 There are no records in the file.

See Also:

31180 NDM_GET_CURRENT_INDEX

' 31410 NDM_SET_POSITION (FILE_HANDLE\$, POSITION_HANDLE\$, RETURN_CODE)

Input Parameters:

FILE_HANDLE\$ - the handle to the file whose position is to be changed.

POSITION_HANDLE\$ - an eight-byte value (returned by **31280 NDM_GET_POSITION**) indicating the position to move to.

Output Parameters:

None

Description:

This function sets the current record pointer of the specified file to the value specified in POSITION_HANDLE\$. POSITION_HANDLE\$ must have been created by a call to **31280 NDM_GET_POSITION**, and cannot have been modified by the application in any way.

This function is typically used only when an application needs to "remember" the location of a particular record and either there is no unique key for the file or the application cannot "remember" the key. To ensure that the record has not been modified by other users, either the record should be locked by the application for the duration of the "get position/set position" operation, or the application must reread the record after the "set position" call.

Pre-Requisites:

- The FILE_HANDLE\$ file must be open.
- The POSITION_HANDLE\$ value must have been returned by a previous call to **31280 NDM_GET_POSITION**.

Currency Changes:

The current record of the file is changed as described.

Toolbox Features:

The "get/set position" toolbox feature (feature # 5) must be set to call this function.

Possible Errors:

3 There are no records in the file.

40 Lost position. The record was deleted or its key value was changed.

42 The record is in use.

See Also:

31280 NDM_GET_POSITION

```
' 31420 NDM_SET_TOOLBOX_STATUS
      ( FEATURE_NUMBER , NEW_STATUS , RETURN_CODE )
```

Input Parameters:

FEATURE_NUMBER - a code identifying the toolbox feature whose status is to be changed.

For a description of each feature refer to Chapter 5 on the toolbox feature. Possible values include:

- 1 Filename extensions
- 2 Key types
- 3 File limits
- 4 Transactions
- 5 Get/Set position
- 6 Create/Delete index

NEW_STATUS - contains one of the following values:

- 0 make the toolbox feature unavailable.
- 1 make the toolbox feature available.

Output Parameters:

None

Description:

This function allows or disallows certain native ISAM specific features.



Any application that calls this function to enable toolbox access WILL NOT be portable across native ISAMs.

The default toolbox status of each feature, set by calling **31320 NDM_INITIALIZE**, is "disabled".

Pre-Requisites:

None

Currency Changes:

None

Toolbox Features:

This function must be called to enable or disable any toolbox feature.

Possible Errors:

None

See Also:

31310 NDM_GET_TOOLBOX_STATUS

' 31425 NDM_SET_TRANSLATION_TABLE (NAME\$, RETURN_CODE)
--

Input Parameters:

NAME\$ - the name of the translation table to be used or "NO TRANSLATION" if no translation is to be performed or if the translation should not change the values being converted.

Output Parameters:

None

Description:

Reads the translation table from the translation table file as specified by NAME\$ and sets the current translation table from the record. If NAME\$ is "NO TRANSLATION" then a record is NOT read from the file. Instead, the current translation table is set so that it will not cause any changes during conversion.

Pre-Requisites:

None

Currency Changes:

None

Toolbox features:

None

Possible Errors:

None

See Also:

31315 NDM_GET_TRANSLATION_TABLE

' 31430 NDM_TRANSACTION_ABORT (RETURN_CODE)
--

Input Parameters:

None

Output Parameters:

None

Description:

This function cancels (undoes) all NDM operations performed since the last **31450 NDM_TRANSACTION_START** call. If this call was not preceded by a matching call to **31450 NDM_TRANSACTION_START**, an error is returned.

NOTE: If the system is shut down abnormally (e.g., power failure, program crash) during a transaction, the next time the system is started up, an implicit **31430 NDM_TRANSACTION_ABORT** call is performed. This protects data files from being in a partially updated state. It is therefore recommended that related file access calls be grouped together in a transaction.

Pre-Requisites:

- Must be preceded by a call to **31450 NDM_TRANSACTION_START**.

Currency Changes:

None

Toolbox Features:

The "transaction" toolbox function (function # 4) must be enabled to call this function.

Possible Errors:

None

See Also:

31440 NDM_TRANSACTION_COMPLETE
31450 NDM_TRANSACTION_START

' 31440 NDM_TRANSACTION_COMPLETE (RETURN_CODE)

Input Parameters:

None

Output Parameters:

None

Description:

This function completes a transaction; makes permanent the changes specified since the last **31450 NDM_TRANSACTION_START** call. If this call was not preceded by a matching **31450 NDM_TRANSACTION_START** call, an error is returned.

Pre-Requisites:

- Must be preceded by a call to **31450 NDM_TRANSACTION_START**.

Currency Changes:

None

Toolbox Features:

The "transaction" toolbox function (function # 4) must be enabled to call this function.

Possible Errors:

None

See Also:

31430 NDM_TRANSACTION_ABORT
31450 NDM_TRANSACTION_START

' 31450 NDM_TRANSACTION_START (RETURN_CODE)
--

Input Parameters:

None

Output Parameters:

None

Description:

This function defines the start of a "transaction", which is a series of API calls that modify one or more data files. A transaction is ended by a call to either **31440 NDM_TRANSACTION_COMPLETE** or **31430 NDM_TRANSACTION_ABORT**.

NOTE: Calling **31450 NDM_TRANSACTION_START** when an unterminated transaction is still in effect constitutes an error.

Pre-Requisites:

None

Currency Changes:

None

Toolbox Features:

The "transaction" toolbox function (function # 4) must be enabled to call this function.

Possible Errors:

None

See Also:

31430 NDM_TRANSACTION_ABORT
31440 NDM_TRANSACTION_COMPLETE

' 31460 NDM_UNLOCK_ALL_RECORDS (FILE_HANDLE\$, RETURN_CODE)
--

Input Parameters:

FILE_HANDLE\$ - the handle to the file containing the record to be unlocked.

Output Parameters:

None

Description:

This function unlocks the record (if any) in the specified file that was locked by a previous call to **31360 NDM_READ_RECORD_BY_KEY**, **31370 NDM_READ_RECORD_BY_POSITION**, **31330 NDM_INSERT_RECORD**, or **31010 NDM_APPEND_UNIQUE_KEY_RECORD** with an "L" in the LOCK\$ parameter.

Pre-Requisites:

- The FILE_HANDLE\$ file must be open.

Currency Changes:

None

Toolbox Features:

None

Possible Errors:

None

See Also:

31010 NDM_APPEND_UNIQUE_KEY_RECORD
31330 NDM_INSERT_RECORDR
31360 NDM_READ_RECORD_BY_KEY
31370 NDM_READ_RECORD_BY_POSITION

CHAPTER 11

RETURN CODES

11.1 Overview

This chapter discusses the various NDM return codes that are used to provide program control for recoverable errors that are generated by the NDM API or the native ISAM.

NOTE: It is essential that the developer provide a method for trapping these codes after each API function call.

Section 11.2 lists all NDM return codes.

Section 11.3 explains the return code file.

Section 11.4 explains each NDM return code in detail and provides help on resolving the difficulty.

11.2 Return Code Listing

This section lists all the NDM return codes.

NOTE: The NDM also keeps track of the most recent native ISAM error code. The application may examine this by calling 31140 NDM_GET_ISAM_RETURN_CODE.

Code	Description
0	No error - the function completed successfully.
1	End of file was encountered.
2	Beginning of file was encountered.
3	File contains no records.
4	I/O error.
5	File is not open.
6	Key value was not found in file.
7	Duplicate keys are not allowed.
8	No index. The "current index" is undefined.
9	No position. The "current record" is undefined.
10	Invalid file name.
11	File was not found.
12	Error while closing file.
13	Disk is full.
14	Unrecoverable error.
15	No access method is active.
16	Key table is too small.
17	Record buffer is too small.
18	File handle is too small.
19	File creation error.
20	Too many key segments are in key description.
21	Key is not in record.
22	Record is too long.
23	Key is too long.
24	File was not created by this access method.
25	Transactions are not enabled.
26	Transaction was already started.
27	Transaction file I/O error.
28	No current transaction.
29	Too many file accesses in transaction.
30	Operation is not allowed inside transaction.

Code	Description
31	Invalid position handle.
32	Invalid key specification (inconsistent key flags).
33	Write is not allowed (file was opened in read-only mode).
34	Too many files are open.
35	Invalid key type.
36	Index file is corrupt.
37	File already exists.
38	Record conflict. Record was changed since last read.
39	Lock table is full.
40	Lost position. Current record was deleted or its key was modified.
41	Record was not read from inside a transaction.
42	Record is in use.
43	File is in use.
44	Invalid open mode.
45	Server device table is full.
46	"Number of concurrent transactions" exceeded.
47	Invalid "LOCK\$" parameter.
48	Permission error. Network privileges disallow file access.
49	Internal transaction error.
50	Invalid page size.
100	Conversion table number not found in DD file.
101	Invalid direction.
102	Invalid seek type.
103	This index can't be deleted.
104	Out of memory.
105	Invalid file handle.
106	Integer parameter expected.
107	Status must be 0 or 1.
108	File name is too long.
109	Record buffer for result of conversion is too small.
110	File deletion failed.
111	Integer parameter is out of range.
112	Invalid conversion length.
113	Transaction log file creation error.
114	Invalid transaction log file.
115	Index number is out of range.
116	Maximum conversion tables are in use.
117	No segments in key.
118	Limits table is too small.
119	Error description buffer is too small.

Code	Description
120	Too many keys.
121	Too many segments in a key.
122	Invalid file type. Unsupported ISAM-specific features are in use.
123	Maximum conversion table entries are in use.
124	ISAM-specific table is too small.
125	Invalid field type in conversion table.
126	Key must have only one segment.
127	Key must be numeric.
128	Name string is too small.
129	Status table is too small.
130	File list buffer is too small.
131	Conversion list buffer is too small.
132	NDM has not been initialized.
133	Invalid feature number.
134	Invalid key in record value.
135	Field name not found in data description file.
136	Toolbox filename extension not activated.
137	Toolbox key types not activated.
138	Toolbox transactions not activated.
139	Toolbox get/set position not activated.
140	Toolbox create/delete index not activated.
141	Security failed.
142	Numeric overflow during conversion.
143	Numeric underflow during conversion.
144	Expecting catalog file.
145	Expecting data description file.
146	Expecting key description file.
147	Invalid conversion table handle.
148	Format specification buffer is too small.
149	Invalid key length.
150	Index numbers are not the same.
999	Unknown error.

11.3 Return Code File

The return code data file contains text messages associated with each NDM return code and each native ISAM return code. This file is a NDM file and can be accessed by the application whenever the application wishes to display information about an unexpected return code.

The text messages in this file may be modified by use of the Error Description File Maintenance Utility (see Chapter 4). This feature is intended to allow translation of the text messages into non-English languages. This capability generally should not be used to otherwise modify the content of the messages.

The return code file contains a record for each return code generated by the native ISAM or the NDM API. The return type field is used to distinguish the NDM codes from the native ISAM codes. A value of 0 is used to indicate NDM and 1 to indicate native ISAM codes.

11.4 Explanation of Return Codes

The return codes are used to provide program control of recoverable errors that are generated by the NDM API. A listing of the text file for these return codes is available through the Error Description File Maintenance Utility. Refer to Chapter 4 of this Programmer's Guide.

NOTE: Return codes should always be checked for by the application. Return codes will not stop program execution.

The following provides a list of the various NDM return codes with an explanation of the problem and suggestions for problem resolution.

0 No error - the function completed successfully.

Explanation: The API function call was executed successfully.

Help: None.

1 End of file was encountered.

Explanation: In an attempt to read from or write to a file, the EOF marker was encountered.

Help: This is a routine error code typically encountered when a program is reading sequentially through a file. Applications should test for this code on all sequential read operations.

2 Beginning of file was encountered.

Explanation: In an attempt to read from or write to a file, the BOF marker was encountered.

Help: This is a routine error code typically encountered when a program is reading sequentially through a file. Applications should test for this code on all sequential read operations.

3 File contains no records.

Explanation: An attempt to read, write, or position in a data file that currently contains no data records has been made.

Help: The file contains no data records or has an invalid data description, key description, conversion table, or index. Verify the correct setting for the data description, key description, conversion table, and index files. If these are set correctly, verify the integrity of the storage disk.

4 I/O error.

Explanation: A physical I/O error occurred during the specified operation. This is typically due to a hardware problem.

Help: Correct the hardware problem.

5 File is not open.

Explanation: An attempt was made to read or manipulate a file that is not currently open.

Help: Before attempting the operation that is generating this return code, open the file in question.

6 Key value was not found in file.

Explanation: The key value that was to be searched for was not found in the data file being addressed.

Help: This error will typically occur with **31360 NDM_READ_RECORD_BY_KEY** operations. Applications should test for this specific error code on any calls to this function.

7 Duplicate keys are not allowed.

Explanation: An attempt was made to enter a new record that matches the key value of an existing record. This index was previously defined as unique and as such duplicate values are not allowed.

Help: The record entry must be modified.

8 No index. The "current index" is undefined.

Explanation: In attempting to read a record, the current index being used is not defined or not defined correctly.

Help: Make sure to specify a valid index when opening the file or use **31400 SET_CURRENT_INDEX** to explicitly establish a valid current index.

9 No position. The "current record" is undefined.

Explanation: While attempting to read or manipulate the current data record, the current record was undefined.

Help: The read has gone past valid data in the file. This is typically caused by reading past the beginning or ending of the data file. The current record must be established by a **31360 READ_BY_KEY** operation or by a **31370 READ_BY_POSITION** operation with a position code of 2 (end of file) or -2 (beginning of file).

10 Invalid file name.

Explanation: The file name being referenced is incorrect.

Help: Check the file name for errors or check the path specified.

11 File was not found.

Explanation: The file name being addressed was not found.

Help: Check the file name for errors or check the path specified.

12 Error while closing file.

Explanation: An error occurred while closing a file. Most often, this occurs when the file being closed is not open.

Help: Make sure this file is the correct file and that it is in fact opened.

13 Disk is full.

Explanation: There is no longer enough room on the storage device to allow the current operation to be completed.

Help: Disk space must be made available for the API function requested to be completed.

14 Unrecoverable error.

Explanation: An error was encountered that is not recoverable.

Help: The native ISAM could not recover from an unknown error.

15 No access method is active.

Explanation: The NDM found no native ISAM active to allow for NDM operation.

Help: Start the native ISAM and then continue the operation.

16 Key table is too small.

Explanation: The key table area was too small to allow for the requested API function.

Help: Increase the size of the key table variable.

17 Record buffer is too small.

Explanation: The record buffer area was too small to allow the requested API function.

Help: Increase the size of the record buffer variable.

18 File handle is too small.

Explanation: The file handle variable was defined to be too small to pass the appropriate value.

Help: Dimension the file handle variable so that all the information being returned fits.

19 File creation error.

Explanation: An error occurred while creating the file.

Help: Check the native ISAM error and correct the condition that caused the error.

20 Too many key segments are in key description.

Explanation: The maximum number of keys and/or segments was exceeded in the key description.

Help: Each key description may contain up to 8 segments. A total of 24 key segments per file may be defined.

21 Key is not in record.

Explanation: Key position falls outside the record.

Help: The start position plus the length of the key must not exceed the record length.

22 Record is too long.

Explanation: The maximum record size supported by the NDM was exceeded.

Help: Decrease the size of the record or specify a larger maximum record length when calling **31320 NDM_INITIALIZE**.

23 Key is too long.

Explanation: The maximum key size of 120 bytes has been exceeded.

Help: Use a smaller key or use the **31420 NDM_SET_TOOLBOX_STATUS** function to allow a larger key size.

24 File was not created by this access method.

Explanation: NDM was asked to open a data file not created or supported by the native ISAM.

Help: Make sure the file being opened is the correct file or provide conversion.

25 Transactions are not enabled.

Explanation: The transaction tracking is an extended feature.

Help: Use **31420 NDM_SET_TOOLBOX_STATUS** to enable it.

26 Transaction was already started.

Explanation: An attempt was made to call **31450 NDM_TRANSACTION_START** while a previous transaction was still in progress.

Help: Allow the first operation to finish before another is attempted.

27 Transaction file I/O error.

Explanation: A physical I/O error occurred during the specified operation. This is typically due to a hardware problem.

Help: Correct the hardware problem.

28 No current transaction.

Explanation: An attempt was made to call **31430 NDM_TRANSACTION_ABORT** or **31440 NDM_TRANSACTION_COMPLETE** without **31450 NDM_TRANSACTION_START**.

Help: Call **31450 NDM_TRANSACTION_START** first.

29 Too many file accesses in transaction.

Explanation: The maximum number of file accesses was exceeded in the current transaction.

Help: Reduce the number of file accesses in the transaction.

30 Operation is not allowed inside transaction.

Explanation: Certain native ISAM operations are not allowed within a transaction.

Help: Check native ISAM documentation for specifics.

31 Invalid position handle.

Explanation: The position pointer is pointing to an invalid data location for the operation being attempted.

Help: Verify the value being returned is the expected value.

32 Invalid key specification (inconsistent key flags).

Explanation: A conflict exists with the way the various key flags are set.

Help: Review the key flags for compatibility and make the appropriate changes.

33 Write is not allowed (file was opened in read-only mode).

Explanation: An attempt was made to write or modify a read-only mode file.

Help: Either this process is not allowed or the file must be set to read or write mode before the API function requested can be completed.

34 Too many files are open.

Explanation: The maximum number of files opened on the system has been exceeded.

Help: Reconfigure the maximum number of open files allowed by the operating system, native ISAM or NDM.

35 Invalid key type.

Explanation: A key type was requested that is not supported by the NDM.

Help: Either the key type must be changed to an NDM supported type or the NDM toolbox feature must be enabled to allow this key type to be used.

36 Index file is corrupt.

Explanation: The index file being used is corrupted.

Help: Attempt to recover the index file and try again. If the index cannot be recovered, it must be rebuilt. The utilities provided with the native ISAM may be helpful.

37 File already exists.

Explanation: The file being created already exists.

Help: If the existing file is to be overwritten, then the existing file must first be deleted by **31110 NDM_DELETE_FILE**.

38 Record conflict. Record was changed since last read.

Explanation: The record being rewritten has been modified by another user since being read by the task performing the rewrite. This error code is only generated on native ISAMs that support passive concurrency. Passive concurrency means that the native ISAM automatically detects the condition where a record being rewritten has been modified by another task. Since not all native ISAMs support this feature, the application should not depend on this feature. Rather, the application should always lock a record that it intends to update. If your application ever generates an error 38, this should be considered an application bug (it did not lock the record it was trying to update) and the program should be corrected.

Help: This can be avoided by locking the record to be rewritten when it is read.

39 Lock table is full.

Explanation: The maximum number of records that can be locked concurrently by the native ISAM has been exceeded.

Help: This parameter can be configured on most native ISAMs.

40 Lost position. Current record was deleted or its key was modified.

Explanation: In attempting to read or manipulate a data file, the position pointer was found invalid.

Help: The position pointer found is probably outside the valid file data.

41 Record was not read from inside a transaction.

Explanation: Btrieve requires that all records modified within a transaction be read within the same transaction.

Help: Change the location in the code where the record is read.

42 Record is in use.

Explanation: The current record is locked and cannot be addressed at this time.

Help: Try reading the record again until it is no longer locked by another user or display a message to the operator.

43 File is in use.

Explanation: The current file is locked and cannot be opened at this time.

Help: Try opening the file again until it is no longer locked by another user or display a message to the operator.

44 Invalid open mode.

Explanation: An attempt was made to open a file in an unsupported mode.

Help: Use a valid mode to open the file.

45 Server device table is full.

Explanation: Btrieve tried to access too many file servers or mapped drives.

Help: Change maximum with BREQUEST program.

46 'Number of concurrent transactions' exceeded.

Explanation: The maximum number of concurrent transactions was exceeded.

Help: Expand allowable number Native ISAM utility program.

47 Invalid 'LOCKS' parameter.

Explanation: An invalid value was specified for the lock parameter.

Help: Check the parameter being used and correct.

48 Permission error. Network privileges disallow file access.

Explanation: The user does not have rights to this file.

Help: The system manager must provide rights to allow access to the files to be used by the NDM.

49 Internal transaction error.

Explanation: Btrieve internal transaction failure on a NetWare TTS file.

Help: Must do a 31430 NDM_TRANSACTION_ABORT.

50 Invalid page size.

Explanation: The page size specified when creating a file was not valid.

Help: Page sizes must be specified in 512 byte increments.

100 Conversion table number not found in DD file.

Explanation: There are no records with the specified conversion table number in the data description file.

Help: Check the conversion table number against the ones in the data description file and change accordingly.

101 Invalid direction.

Explanation: The search direction parameter contained an illegal value.

Help: Check the search parameters and correct the invalid entry.

102 Invalid seek type.

Explanation: The seek type requested is not allowed under the NDM.

Help: Check the seek type and correct the invalid entry.

103 This index can't be deleted.

Explanation: The index may not be deleted.

Help: This is either a primary index or the user does not have rights to delete this index.

104 Out of memory.

Explanation: There is not enough memory to perform the requested API function.

Help: Memory must be made available to allow the API function to continue.

105 Invalid file handle.

Explanation: The handle specified does not point to a valid file handle.

Help: Check the file handle value and correct.

106 Integer parameter expected.

Explanation: The native ISAM was expecting an integer value.

Help: Revise the parameter to an integer or use the NDM conversion features to make this conversion.

107 Status must be 0 or 1.

Explanation: SET_TOOL_BOX_STATUS status parameter must be 0 or 1.

Help: Check the value passed.

108 File name is too long.

Explanation: The length of the file name was too long for the native operating system.

Help: Reduce the size of the file name to one that is acceptable under the host operating system.

109 Record buffer for result of conversion is too small.

Explanation: The size of the conversion table is too small to allow for the requested field type conversions.

Help: Increase the size of the record buffer.

110 File deletion failed.

Explanation: The file was not successfully deleted.

Help: The user must have rights to allow for the deletion of the file.

111 Integer parameter is out of range.

Explanation: An integer parameter value was greater than 32,767 or less than -32,768.

Help: Check the value of the passed parameter and verify its accuracy.

112 Invalid conversion length.

Explanation: The length of a conversion data item is incorrect.

Help: Check against lengths in Field type file.

113 Transaction log file creation error.

Explanation: Error trying to create native ISAM transaction log file.

Help: Ensure adequate disk space and file creation privileges.

114 Invalid transaction log file.

Explanation: Cannot read native ISAM log file or log file record format is invalid.

Help: Ensure transaction log file has not been accidentally overwritten.

115 Index number is out of range.

Explanation: An attempt was made to use an index number that does not exist.

Help: Check the index number or enable the toolbox feature using **31420 NDM_SET_TOOLBOX_STATUS**, if using extended features.

116 Maximum conversion tables are in use.

Explanation: An attempt was made to call **31050 NDM_CREATE_CONVERSION_TABLE** when all tables are in use.

Help: Either expand the number of conversion tables specified to **31320 NDM_INITIALIZE** or delete the conversion tables that are no longer required.

117 No segments in key.

Explanation: No segments were found in the key being addressed.

Help: There must be at least 1 segment in a key.

118 Limits table is too small.

Explanation: The limits table must be at least 16 bytes long for **31260 GET_LIMIT_HIGHWATERS**.

Help: Check the length.

119 Error description buffer is too small.

Explanation: The space allocated to hold the return code description was too small.

Help: Increase the size of the return description buffer to hold the longest possible return code description.

120 Too many keys.

Explanation: The number of keys supported by the NDM has been exceeded.

Help: Reduce the number of keys in use or use **31420 NDM_SET_TOOLBOX_STATUS** to allow the use of more keys by enabling the "file limits" toolbox feature, option # 3.

121 Too many segments in a key.

Explanation: The number of key segments supported by the native ISAM has been exceeded.

Help: Refer to the native ISAM manual for information on the maximum number of key segments supported and then make the appropriate changes in the NDM, or use **31420 NDM_SET_TOOLBOX_STATUS** to allow the use of more keys by enabling the "file limits" feature, option # 3.

122 Invalid file type. Unsupported ISAM-specific features are in use.

Explanation: An attempt was made to use a feature not supported by the native ISAM.

Help: This feature cannot be used with the NDM. Another solution must be found for the required API function.

123 Maximum conversion table entries are in use.

Explanation: An attempt was made to call **31050 NDM_CREATE_CONVERSION_TABLE** when all table entries are in use.

Help: Either expand the number of conversion table entries specified to **31320 NDM_INITIALIZE** or delete the conversion tables that are no longer required.

124 ISAM-specific buffer is too small.

Explanation: The native ISAM specific buffer must be at least 128 bytes long.

Help: Check the size and increase.

125 Invalid field type in conversion table.

Explanation: An unsupported field type was entered in the conversion table.

Help: Verify the field type being passed to the NDM conversion routines.

126 Key must have only one segment.

Explanation: The key to be incremented for **31010 APPEND_UNIQUE_KEY_RECORD** must have only one segment.

Help: Check that the correct key is being used.

127 Key must be numeric.

Explanation: The key to be incremented for **31010 APPEND_UNIQUE_KEY_RECORD** must be a numeric type.

Help: Check that the correct key is being used.

128 Name string is too small.

Explanation: The name string is too small for the operation being attempted.

Help: Increase the size allocated for the name string or decrease the size of the name.

129 Status table is too small.

Explanation: The status table is too small for the operation being attempted.

Help: Increase the size of the status table.

130 File list buffer is too small.

Explanation: The file list buffer is too small for the operation being attempted.

Help: Increase the size of the file list buffer.

131 Conversion list buffer is too small.

Explanation: The conversion list buffer is too small for the operation being attempted.

Help: Increase the size of the conversion list buffer.

132 NDM has not been initialized.

Explanation: The NDM has not been initialized.

Help: **31320 NDM_INITIALIZE** must be called before doing anything else.

133 Invalid feature number.

Explanation: The toolbox feature number is not valid.

Help: Check and correct.

134 Invalid key in record value.

Explanation: The **KEY_IN_RECORD** parameter in **31040 NDM_CREATE_CONV_TABLE_FOR_KEY** must be Y or N.

Help: Check and correct.

135 Field name not found in data description file.

Explanation: A field name specified in the name list for **31050 CREATE_CONVERSION_TABLE** could not be found in the data description file.

Help: Verify that all names are in the data description file.

136 Toolbox filename extension not activated.

Explanation: A filename extension was used when the toolbox feature was not active.

Help: Either do not use an extension on the filename, or enable the toolbox feature.

137 Toolbox key types not activated.

Explanation: An extended key type was used when the toolbox feature was not active.

Help: Either do not use the extended key type, or enable the toolbox feature.

138 Toolbox transactions not activated.

Explanation: A transaction was started when the toolbox feature was not active.

Help: Either do not use the transaction, or enable the toolbox feature.

139 Toolbox get/set position not activated.

Explanation: A get or set position operation was used when the toolbox feature was not active.

Help: Either do not use these operations, or enable the toolbox feature.

140 Toolbox create/delete index not activated.

Explanation: A create or delete index operation was used when the toolbox feature was not active.

Help: Either do not use these operations, or enable the toolbox feature.

141 Security failed.

Explanation: The NDM security check failed during initialization.

Help: The system was not properly installed or the security files have been corrupted.

142 Numeric overflow during conversion.

Explanation: A numeric value is too large for the data type that it is being converted to.

Help: Verify that the data types are compatible.

143 Numeric underflow during conversion.

Explanation: A numeric value is too small for the data type that it is being converted to.

Help: Verify that the data types are compatible.

144 Expecting catalog file.

Explanation: A catalog file handle parameter was expected.

Help: Check that the correct file handle is being used.

145 Expecting data description file.

Explanation: A data description file handle parameter was expected.

Help: Check that the correct file handle is being used.

146 Expecting key description file.

Explanation: A key description file handle parameter was expected.

Help: Check that the correct file handle is being used.

147 Invalid conversion table handle.

Explanation: The handle specified does not point to a valid conversion table.

Help: Check the conversion table handle and correct.

148 Format specification buffer is too small.

Explanation: The space allocated to hold the format specification was too small.

Help: Increase the size of the format specification buffer to hold the largest possible specification.

149 Invalid key length.

Explanation: The length of a specific segment does not correspond correctly with the type of the key segment.

Help: Change the length of the key segment to correspond with the lengths given for the key type in the field type file.

150 Index numbers are not the same.

Explanation: The index numbers in the key description table passed to **31080 NDM_CREATE_INDEX** are not all the same.

Help: Ensure that all of the index numbers are the same and that the list of segments is terminated by an index number of 0.

999 Unknown error.

Explanation: The return code generated is not a valid return code.

Help: Contact Niakwa.

APPENDIX A

DATA DICTIONARY FILE CREATION TUTORIAL

L.1 Overview

The following provides a step-by-step example of creating all necessary data dictionary files that are essential for NDM applications. This example is based on the example program provided with the NDM and discussed in Chapter 5 of this Programmer's Guide. Also provided in this section is an example detailing the steps involved in converting the NDM data dictionary files into IQ data dictionary files.

This example can be viewed as a guideline for creating the necessary data dictionary files for any NDM applications.

The following files will be created by this example.

TESTCAT	The example program's catalog file.
TESTDD	The example program's data description file.
TESTKD	The example program's key description file.
TESTREL	The example program's relation file.
DDMASTER	The IQ data dictionary (optional, for use with IQ only).

For the purposes of this example, the file names TESTxxx have been used for the data dictionary files. The actual data dictionary file names used in Chapter 6 are, CUSTxxx. This name change is made because the data dictionary files used in Chapter 6 are already included on the distribution disks, and the NDM Utilities would not allow their creation.

Section A.2 discusses the guidelines for creating NDM data dictionary files.

Section A.3 discusses the creation of the catalog file.

Section A.4 discusses the creation of the data description file.

Section A.5 discusses the creation of the key description file.

Section A.6 discusses creating an entry in the catalog file for the data file.

Section A.7 discusses adding and modifying data in the data file.

Section A.8 discusses converting the NDM data dictionary files to IQ.

L.2 Data Dictionary File Creation Guidelines

To properly create a data file and its associated catalog file entry, data description file, relation file and key description file, the files must be created in a specific order. This order is as follows:

1. Create the new catalog file.

NOTE: A new catalog file typically is created only once for an entire application and not for each data file.

2. Create the relation file associated with the new catalog file. This is done implicitly by the NDM Utilities during creation of the catalog file.
3. Create the new data description file.
4. Create the new key description file.
5. Create an entry in the current catalog file for the new data file.

NOTE: The utilities that create the data description, key description and relation files, automatically create the necessary entries in the catalog file. As such, it is only necessary for the operator to create an entry for the data file.

6. Optionally, create a new data file using the Access Catalog File Utility.
7. Optionally, convert the NDM data dictionary files into an IQ data dictionary file, if IQ is to be used.

L.3 Creating the Catalog File

To create the new catalog file TESTCAT.DAT, follow the steps shown below.

1. Select the Install New Catalog File Utility from the Utilities Main Menu.
2. When prompted, as shown in Figure A-1, enter **TESTCAT** as the catalog file name, **TESTREL** for the relation file name, and leave blank the from catalog file name entry, (the default **NDMCAT** will be used as source). After pressing Execute, this Utility creates the catalog file and relation file. The required NDM system and support file entries are then copied into the TESTCAT file from the NDMCAT file.

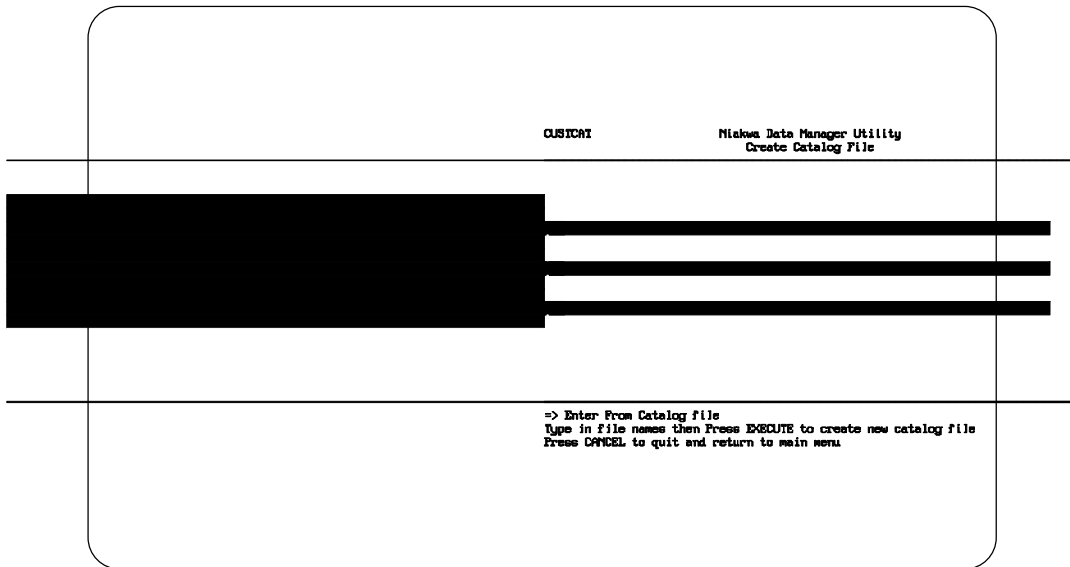


Figure L - 1

NOTE: The first time a new catalog file is created the source catalog file will be NDMCAT. After a new catalog file has been created, the operator has the choice of selecting any other catalog file or leave the from catalog file name entry blank, (in order to use the NDMCAT as the source catalog).

This Utility sets the current catalog file to the name of the catalog file created. The name of the new catalog file will appear in the upper left hand corner of the NDMUTIL main screen. This catalog file name is saved by the NDM Utilities and will appear as the default catalog each time the NDM Utilities is executed. The operator can change the catalog file name displayed by selecting another catalog file with the Set NDMUTIL Option.

L.4 Creating the Data Description File

Once the TESTCAT file is created, TESTDD, the sample data description file can be created by using the Access Data Description File Utility as shown below.

1. Select the Access Data Description File Utility from the Utilities Main Menu.

2. Press Insert to create a new data description file.
3. A window appears prompting the operator for the file title. Enter **TESTDD** and press Enter. Refer to Figure A-2.
4. Enter **TESTDD** for the file name and press Enter (this assumes that TESTDD is to be created in the current directory). Refer to Figure A-2.
5. Press Execute.

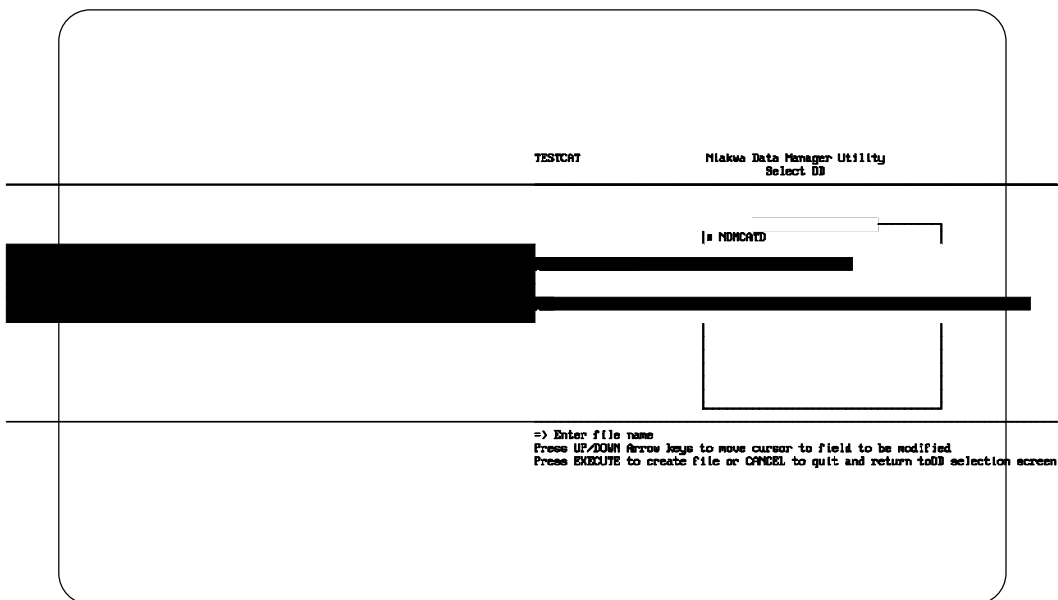


Figure L - 2

6. A window then appears prompting the operator to select a data description file to be used as a source file for the new data description (this allows easier updating of existing data description files). In this example, we want to create a completely new data description file, so there is no need for a source file. As such, press Cancel to exit this window and continue to the next step.
7. A screen appears, as is shown in Figure A-3, allowing the operator to begin the entry of the new data description information. Enter the information to describe the various fields for the data file. The information for the first field, CUSTOMER NUMBER is shown below. Enter this information in the fields provided.

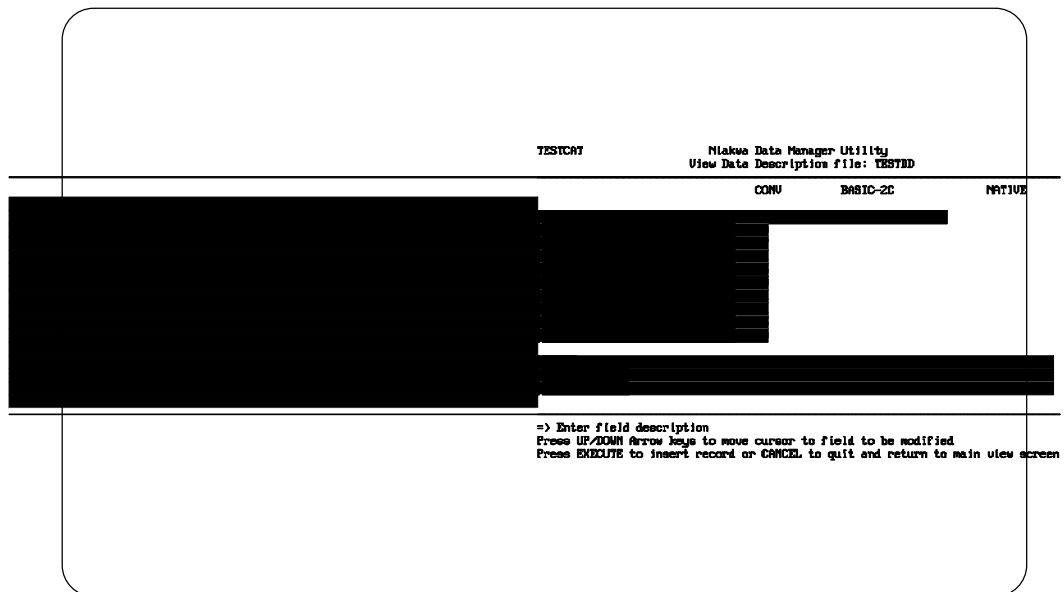


Figure L - 3

8. Press Execute to insert the record.
9. The Utility remains in INSERT mode, to allow entry of additional fields.
10. The entries for the remaining data fields are shown below in Figure A-4. Enter this data for the remaining fields in the same manner that CUSTOMER NUMBER was entered above.

TESTCAT		Makew Data Manager Utility View Data Description file: TEST00									
FIELD NAME	CONV	TABLE	TYPE	START	LEN	DEC	TABLE	START	LEN	DEC	
CUSTOMER NUMBER	0	7	1	5	0		1	1	5	0	
NAME	0	7	6	20	0		1	6	20	0	
PHONE	0	7	26	13	0		1	26	13	0	
VTD SALES\$	0	5	39	5	0		9	39	5	0	
NUMBER OF SALES	0	6	44	2	0		2	44	2	0	
CITY	0	7	46	20	0		1	46	20	0	
NAME-1	1	7	6	5	0		1	6	5	0	
AREA-CODE	2	7	27	3	0		1	27	3	0	

Press FREQ/NEXT to view file, UP/DOWN Arrow to select record, TAB to select index
 EXECUTE/INSERT/DELETE keys to view-modify/insert/delete selected record
 Press F to print file, R to reorg file, Cancel key to return to main menu.

Figure L - 5

12. Press Cancel to return to the Utilities Main Menu.

L.5 Creating the Key Description File

After the data description file is created, the key description file can be created.

To create the TESTKD key description file, follow the steps shown below.

1. Select the Access Key Description File Utility from the Utilities Main Menu.
2. Press Insert to create a new key description file.
3. The Utility prompts the operator for the file title of the key description file. Enter **TESTKD** and press Enter. Refer to Figure A-6.
4. Enter **TESTKD** for the file name and press Enter (this assumes that TESTKD is in the current directory). Refer to Figure A-6.

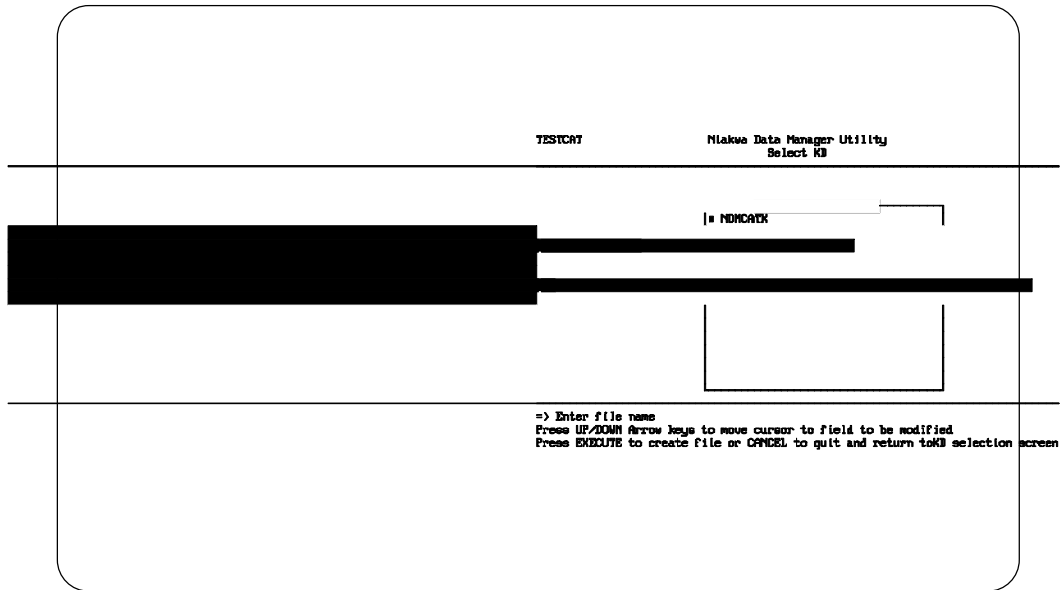


Figure L - 6

5. Press Execute to update.
6. A window then appears prompting the operator to select a key description file to be used as a source file for the new key description file (this allows easier updating of key description files). In this example, we want to create a completely new key description file, so there is no need for a source file. As such, press Cancel to exit this window and continue to the next step.
7. Next, a window appears prompting the operator to select the name of the related data description file. Select **TESTDD** and press Execute. This creates the relation between the data description and key description file and stores this in the TESTREL file created in Section A.3 of this Appendix.
8. Enter the information for the first index to be created. A sample screen is shown in Figure A-7 with the information necessary for segment 0 of the first index to be created.

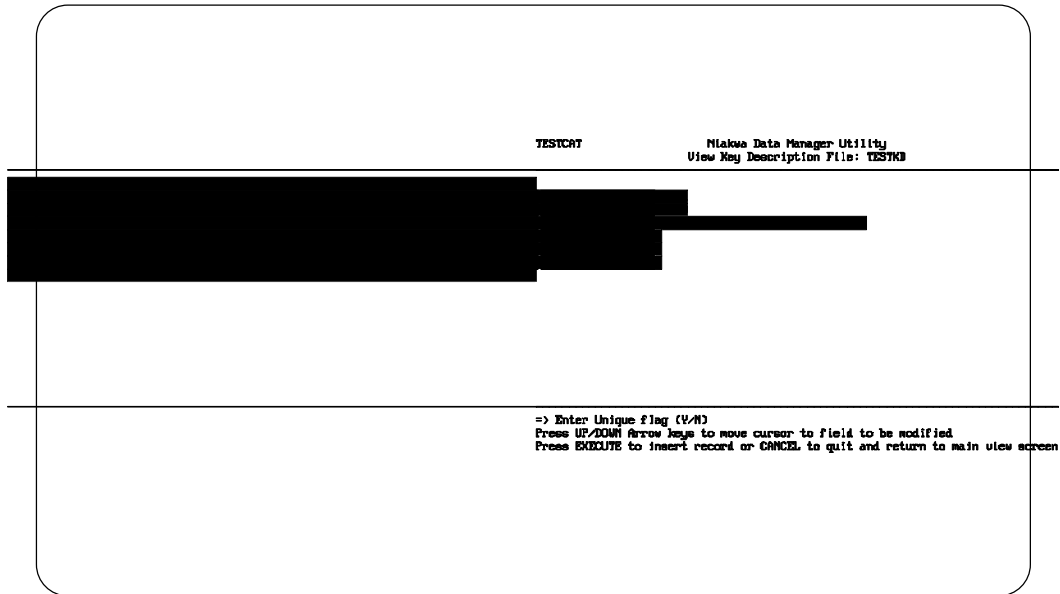


Figure L - 7

A "Y" in the KEY COMPRESSION field allows the key to be stored as a compressed field.

NOTE: For each index created, at least 2 segments are required. The first, segment 0, provides a mnemonic name for the index. When entering information for segment 0, the operator is not prompted for the Ascending field, only for the Unique field. Uniqueness is described only in segment 0. Segment 1 is the first actual segment of every index.

Once the operator enters a number other than 0 for a segment number, the field name IndexName changes to Field Name. When the operator is entering information for the Field Name, a window displays with the field names from the related data description. The operator should then select the appropriate field name for this segment from this window.

NOTE: When entering information on a non-zero segment, the Utility allows the operator to specify the Ascending field.

9. Enter the information for segment 0 and press Execute.

10. The Utility stays in INSERT mode to allow the entry of any additional key description file information.
11. The data for the remaining segments of index 1 and the other indices are shown in Figure A-8. Enter the following indices. After entering each segment press Execute to update the key description file and continue with the next segment.

TESTDAT Minkwa Data Manager Utility
View Key Description File: TESTKD

FIELD NAME	ASC?	UNIQUE	INDEX	SEGMENT	COMP?
■ CUST #		V	1	0	
CUSTOMER NUMBER	V		1	1	
SORT NAME		N	2	0	
NAME-1	V		2	1	
SALES BY AREA		N	3	0	
AREA-CODE	V		3	1	
YTD SALES\$		N	3	2	

Press F8/PAGE/NEXT to view file, UP/DOWN Arrow to select record, TAB to select index
EXECUTE/INSERT/DELETE keys to view-modify/insert/delete selected record
Press F to print file, CANCEL key to return to main menu

Figure L - 8

12. When all segments have been entered press Cancel to return to the Main View Key Description Screen as shown in Figure A-9.

TESTCAT		Nakwa Data Manager Utility View Key Description File: TESTMD			
FIELD NAME	ASC?	UNIQUE	INDEX	SEGMENT	COMP?
# CUST #		V	V	1	0
CUSTOMER NUMBER	V			1	1
SORT NAME		N	N	2	0
NAME-1	V			2	1
SALES BY AREA		N	N	3	0
AREA-CODE	V			3	1
VTD SALES\$	N			3	2

Press FREQ/NEXT to view file, UP/DOWN Arrow to select record, TAB to select index
 EXECUTE/INSERT/DELETE keys to view/modify/insert/delete selected record
 Press F to print file, CANCEL key to return to main menu

Figure L - 9

13. To return to the Utilities Main Menu, press Cancel.

L.6 Creating a Catalog Entry for the Data File

This section shows how to create the catalog file entry for the example native ISAM data file, TEST.

To create this entry, follow the steps shown below.

1. Select the Access Catalog File Utility from the Utilities Main Menu.
2. The screen shown in Figure A-10 appears:

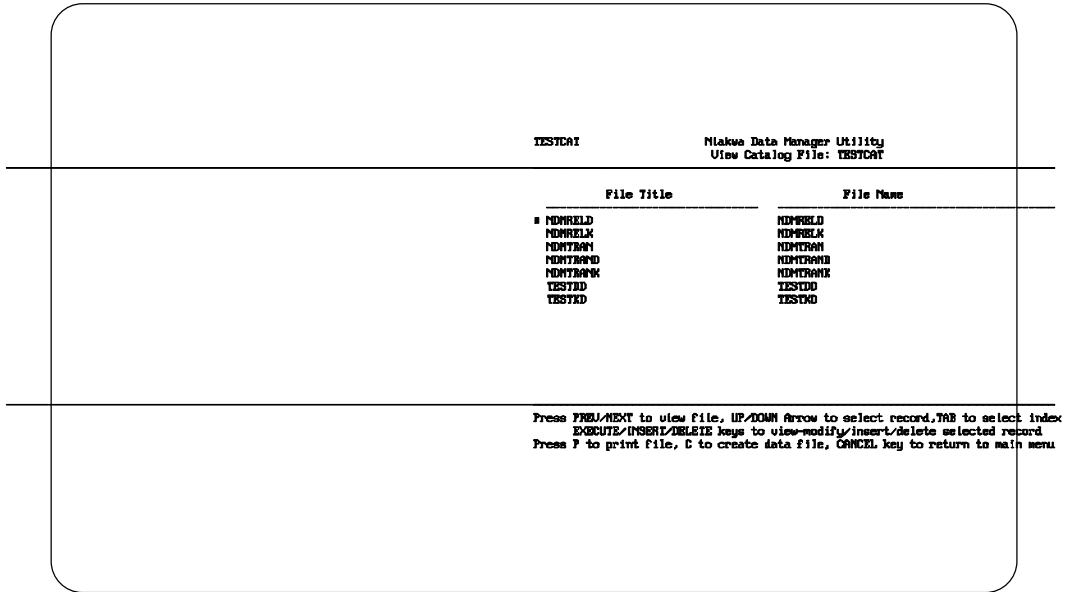


Figure L - 10

Notice the data description and key description files already have catalog entries. These entries were created by the NDM Utilities.

4. Press Insert to create the catalog entry for the data file, TEST.
5. The window shown in Figure A-11 then appears.

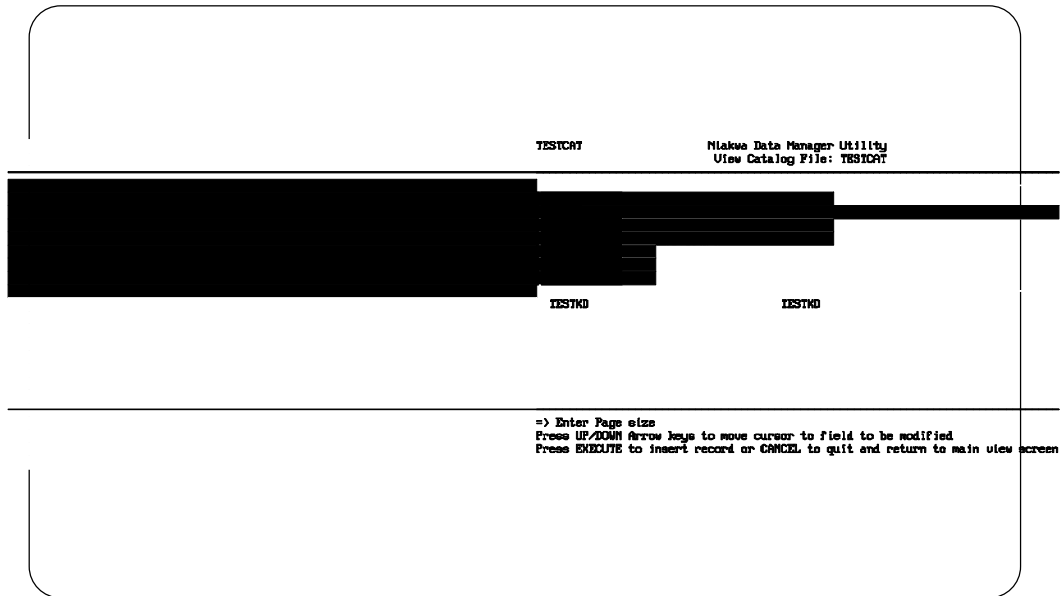


Figure L - 11

Enter TEST in the file title field. Enter the physical location for the file in the file name field. For the data description and key description file entries, a window of valid data description and key description file titles appears. To select TESTDD and TESTKD respectively, press Tab to choose the window and then select the file name by using the Arrow Keys. Press Execute to select the file name.

NOTE: Under Btrieve the Page Size should be 1024 and the Pre-Allocation size is 0.

6. Press Execute to create the new entry.
7. The Utility returns to the Main View Screen.
8. Select the new data file entry with the use of the Arrow keys.
9. Press C, to create the new data file. A window appears as in Figure A-12, to confirm the choice to create the file. Press Enter to create the file.

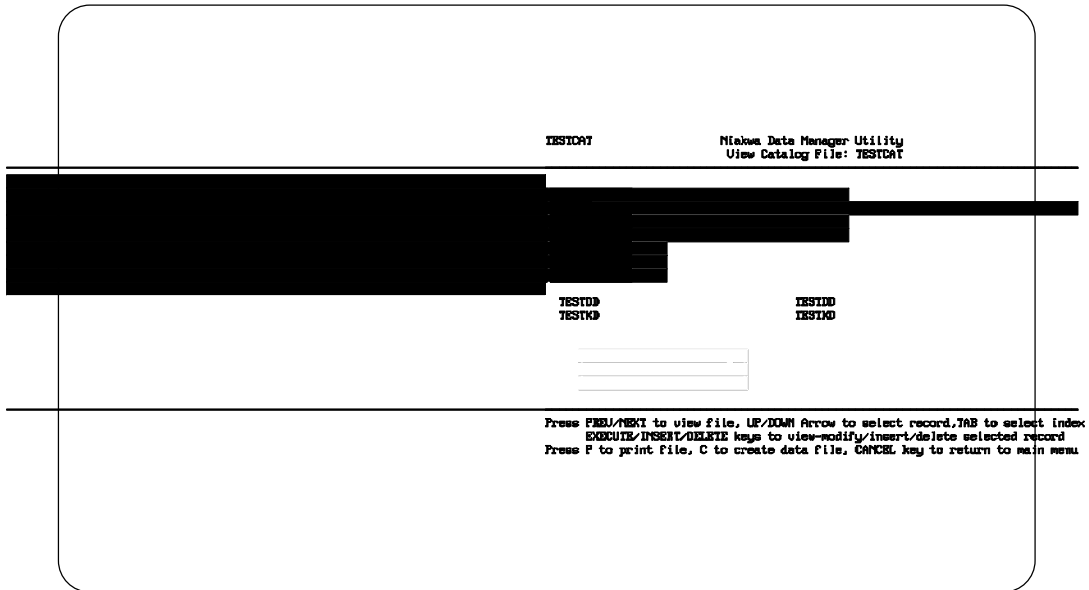


Figure L - 12

10. Answer "Y" to create the data file even if extended key type were used (this file will not be necessarily be portable).
11. Press Cancel to return the Utilities Main Menu.

L.7 Adding/Modifying Data File Records

This section explains how to add or modify information within a data file using the NDM Utilities. Refer to Section A.6 for details on how to create a data file, if one has not yet been created. To add or modify information in a data file, first select the Access User Data File Utility from the Utilities Main Menu and follow the steps listed below.

NOTE: No provisions are made to verify the data entered in this manner.

1. After selecting this Utility, the window shown in Figure A-13 appears:

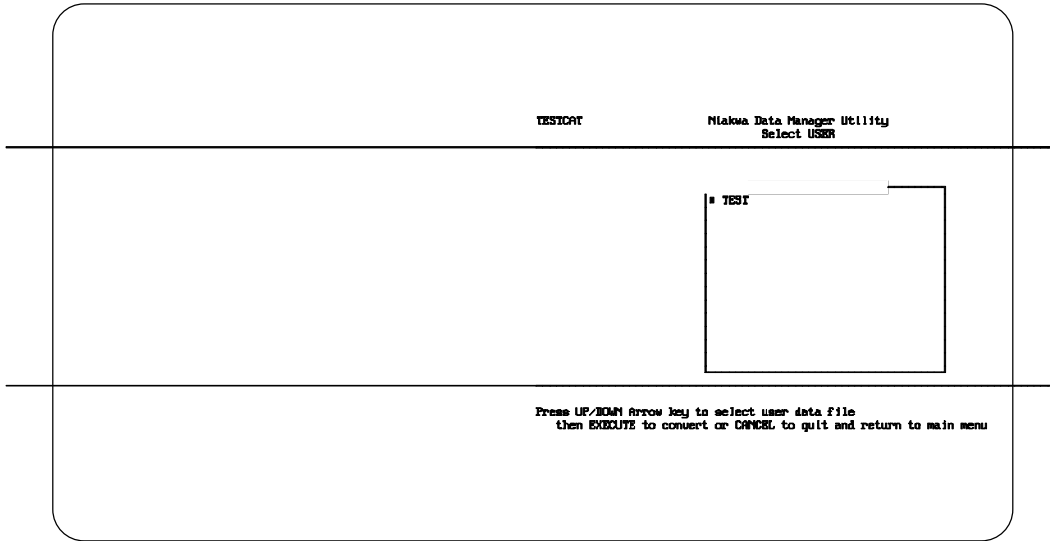


Figure L - 13

2. Select the TEST data file and press Execute. Once a selection is made the Utility displays a data entry window as shown in Figure A-14.

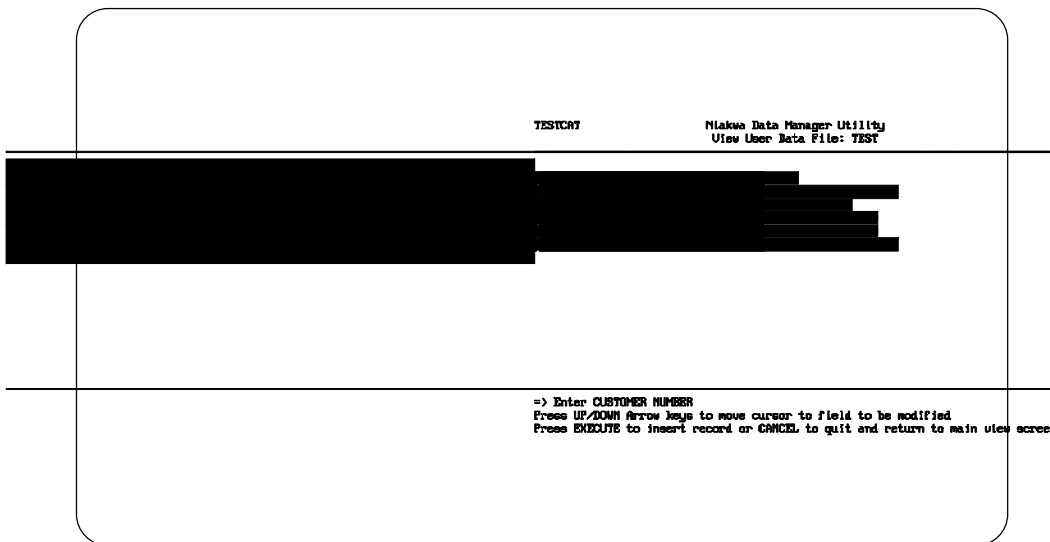


Figure L - 14

3. Enter the information shown in Figure A-15 in the fields provided:

The screenshot shows a terminal window titled 'Make Data Manager Utility' with the subtitle 'View User Data File: TEST'. The screen is divided into two horizontal sections by a line. The top section contains a data entry form with several fields, all of which are redacted with black boxes. The bottom section contains instructions for navigating the utility:

```
TESTCMT          Make Data Manager Utility
                  View User Data File: TEST

=> Enter CITY
Press UP/DOWN Arrow keys to move cursor to field to be modified
Press EXECUTE to insert record or CANCEL to quit and return to main view screen
```

Figure L - 15

4. Press Execute to insert the record. The Utility will remain in INSERT mode and allow the entry of additional records.
5. For the next record enter the information shown in Figure A-16.

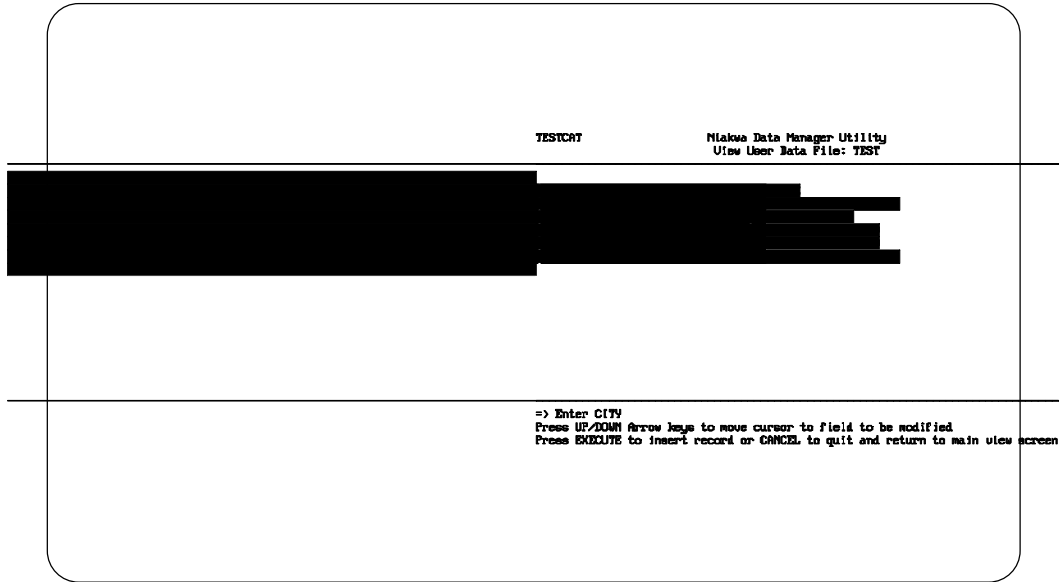


Figure L - 16

6. Press Execute to save and update, then press Cancel to exit INSERT mode and return to the View User Data Screen.
7. The screen shown in Figure A-17 then appears:

TESTCAT		Nakva Data Manager Utility View User Data File: TEST		
CUSTOMER NUMBER	NAME	PHONE	YTD SALES\$	
60002	JOHN FREBLR	(312)123-0988	1000	
60100	JOHN SMITH	(708)555-1212	10000	

Press FREQ/NEXT to view file, UP/DOWN arrow to select record, TAB to select index
 EXECUTE/INSERT/DELETE keys to view/modify/insert/delete selected record
 Press F to print file, CANCEL key to return to main menu

Figure L - 17

9. Press Cancel to return to the Utilities Main Menu.

L.8 Converting an NDM Data Dictionary to IQ

This section describes how to create an IQ data dictionary file for the data file just created. This Utility automatically creates an IQ data dictionary from the NDM data dictionary files TESTDD and TESTKD.

To create the IQ data dictionary file DDMASTER, follow the steps shown below.

1. Select the Export Data Dictionary to IQ Utility from the Utilities Main Menu.
2. The window shown in Figure A-18 appears, with the names of data files associated with the current catalog file, TESTCAT. In this case, only one data file should exist.

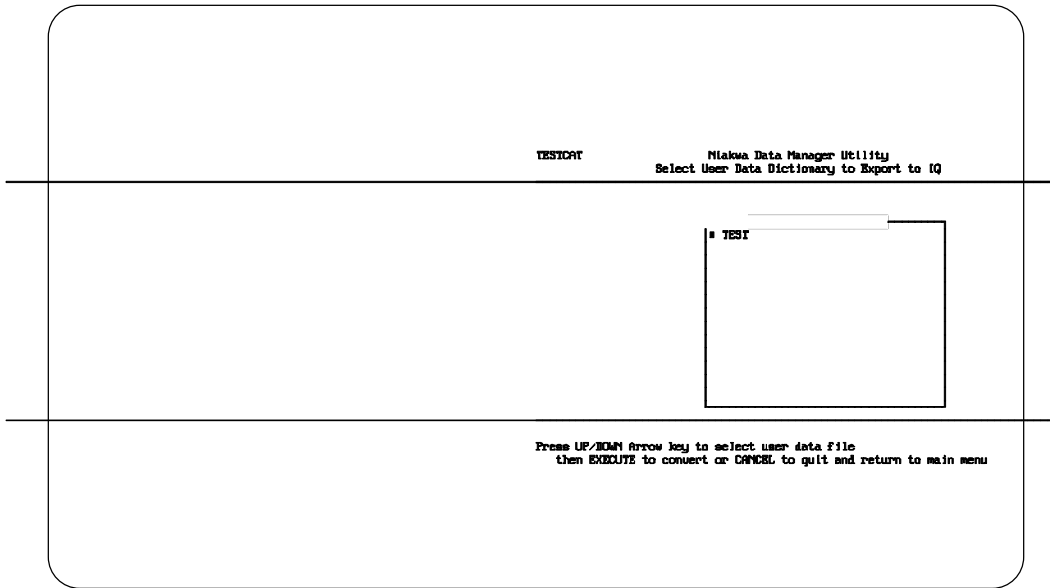


Figure L - 18

3. Select TEST and press Execute.
4. The window shown in Figure A-19 then appears:

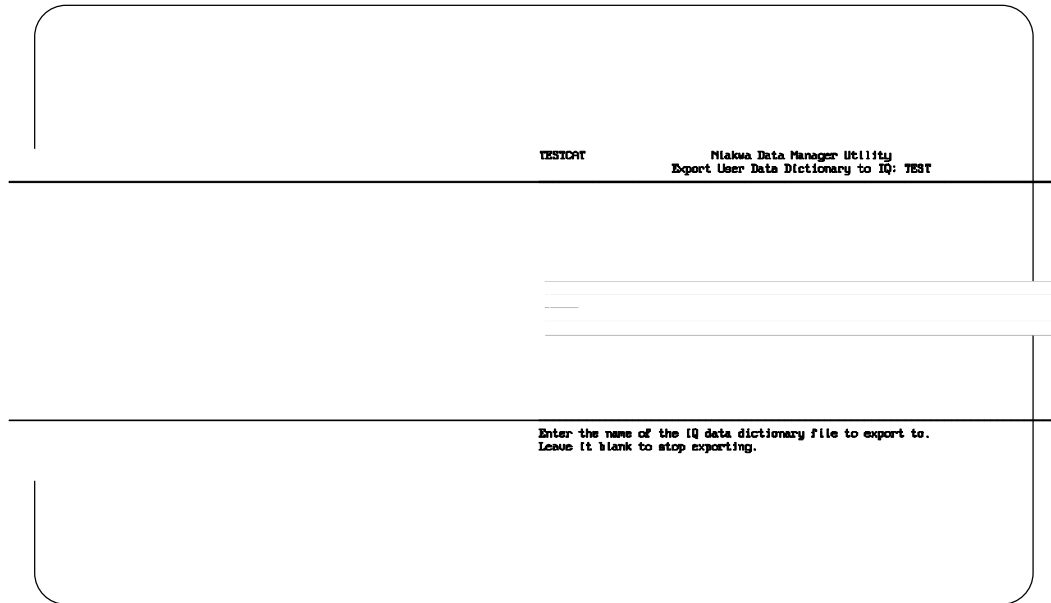


Figure L - 19

5. Accept the default choice, DDMASTER.DAT as the name of the IQ data dictionary by pressing Enter.
6. The window in Figure A-20 then appears:

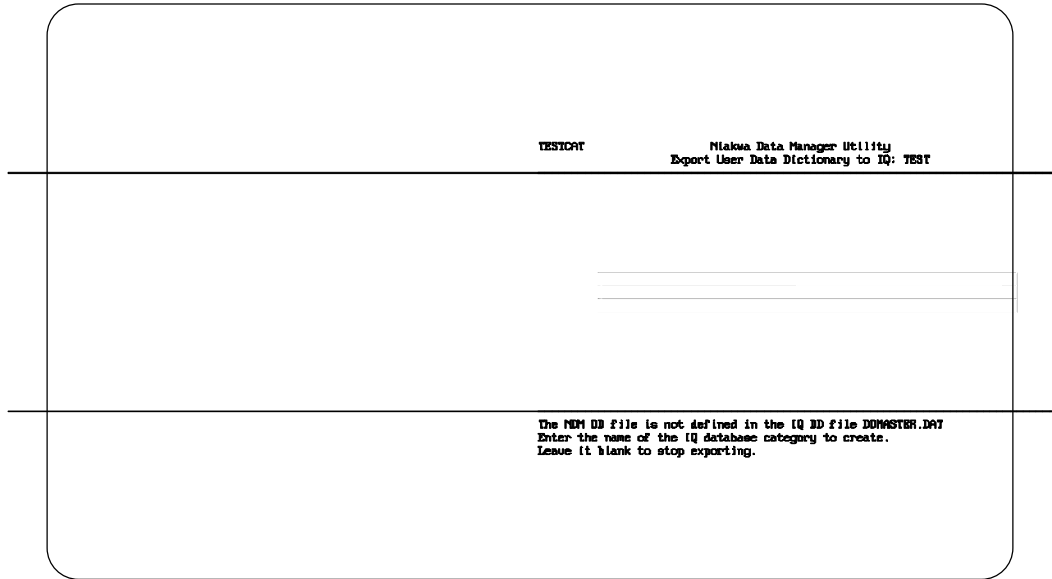


Figure L - 20

Enter TEST for the IQ database category name, and press Execute. The export process will begin and upon completion the select data file window reappears.

The IQ program can now be used to select and display data from the data file TEST.

NOTE: It may be necessary to copy DDMASTER.DAT into the IQ directory if the IQ environment variable is not set correctly. Refer to the IQ documentation for information on the IQ environment variable.

This concludes the creation of all necessary NDM data dictionary files for the example program in Chapter 9 of this Programmer’s Guide, as well as the creation of the native ISAM data file and the associated IQ data dictionary.

APPENDIX B

NDM INTERNAL TABLES

B.1 Overview

The following section lists the NDM internal tables.

Section B.1 describes the NDM Key Description Table format.

Section B.2 describes the Limits Table format.

Section B.3 describes the Native ISAM Table format.

B.2 NDM Key Description Table Format

Table B-1 defines the internal structure of the key description table used to describe the segments of a key.

It is in the format required by the **31060 CREATE_FILE** function.

Start	Length	Type	Description
1	2	N	Segment start position
3	2	N	Segment length
5	1	N	Segment type (native)
6	1	A	Flags
7	1	N	Index number
8	1	A	Key compression
9	6	A	Reserved for future expansion

Table B-1

The "flags" field is defined in Table B-2.

Bits	Meaning
0	Duplicates are allowed
1	Descending order

Table B-2

The numeric values in the key description table are in Basic-2C format, unsigned binary (B00x), where x is 1 for one-byte fields or 2 for two-byte fields.

The maximum number of segments per data file is fixed at 24. This is because the segment information is stored in the file handle which is of fixed size.

B.3 Limits Table

This structure is used by **31260 NDM_GET_LIMIT_HIGHWATERS** to describe the capacities of NDM. Each entry is a 4-byte integer, which can be stored with BIN(N,4).

Item	Name	Description
1	NDM_RECLEN	Maximum data file record length in bytes.
2	NDM_KEYLEN	Maximum length of one key in bytes.
3	NDM_KEYCOUNT	Maximum number of keys per file.
4	NDM_SEGKEY	Maximum number of segments per key.
5-16		Reserved for future expansion.

Table B-3

B.4 Native ISAM Specific Table Format

Table B-4 describes what is used by the ISAM_SPECIFIC\$ parameter of **31060 NDM_CREATE_FILE** to allow the use of native ISAM specific features that do not affect the design of the application. The first byte of this field identifies the native ISAM in use. If this value is not correct, none of the other values in this structure will be used, and the corresponding parameters will be given default values.

Start	Length	Type	Sub-field Name
1	1	N	Native ISAM code (must be a HEX(01) for Btrieve)
2	2	N	Data file page size in bytes (for Btrieve)
4	2	N	Number of Pre-allocation pages (for Btrieve)
6	123	A	Reserved for future expansion

Table B-4

GLOSSARY

Accessee File

The "many" in a one-to-many relationship. If the one-to-one flag is set, there is only one accessee record for each accessor record.

Accessee Index

Refers to the index name given in segment zero of the desired index of the accessee file.

Accessor File

The "one" in a one-to-many relationship, with the accessee the many. If the one-to-one flag is set, there is only one accessee record for each accessor record.

Accessor Index

Used to describe the position of the key in the accessee record to SEEK the accessor file.

API

Application Program Interface.

BOF

Beginning Of File.

Catalog File

The catalog file is used to maintain information about logical file titles. The catalog file contains one record for each file used by the application. This record contains the logical file title, physical file location, and file titles for the key description and data description files.

Data Dictionary

A data dictionary is a group of NDM files that describe NDM data files. These include the catalog, data description, key description and relation files. These are stored as native ISAM files, and therefore can be read and modified by normal NDM file access routines. The NDM provides utilities to maintain each type of data dictionary file. Refer to Chapter 3 and Chapter 4 for more information.

Data Dictionary System Files

Data dictionary system files are used by the NDM Utilities to maintain NDM data dictionary files. Each type of NDM data dictionary file has two associated data dictionary system files: a data description file and a key description file.

Data File

A data file is a collection of related records treated as a unit. A customer mailing list or a customer account list is an example of a data file.

Data Description File

A data description file is used to describe the fields and sub-fields of the data record and to define all conversion information for the records of this file. Each NDM data file has an associated data description file. Refer to Chapter 3 for more information.

Data Description System File

The data description file that describes the structure of all data description files.

EOF

End Of File.

Field

A field is a defined part of a record. It consists of a group of characters that constitute an item of information. Examples of fields within records are an address field or a name field.

Field Name

The name of a field.

Field Type

The type of field (i.e. alpha, packed decimal, etc.).

File Handle

The file handle contains information about a currently open file that is required for any NDM access to the file. File handles may be stored by NDM or by the application depending on parameters passed to 31320 NDM_INITIALIZE.

Index

An index is an ordered set of key values associated with records in a file. The NDM uses indices to maintain a sorted order of records in a file. A file can have more than one index. Refer to Chapter 3 for more information.

Key

A key is a group of characters used to identify or locate a record. The NDM uses keys to identify specific records in a file. By using a key, the NDM can efficiently select the record from the entire collection of records in a file. A key can contain multiple segments. That is to say, a key can be made up of more than one field. Segments do not have to be contiguous. Refer to Chapter 3 for more information.

Key Description File

A key description file describes all the indices of a data file. A data file's key description file contains one record for each segment of each index in the data file. Refer to Chapter 4 for more information.

Native ISAM

A third-party indexed sequential access method such as Btrieve, Sdtrieve or C-ISAM.

NDM Environment Variable

A variable used to define the location of the NDM files on the host system.

Record

A record is a set of one or more consecutive fields on a related subject, for example, a customer in a customer mailing list. A record represents a set of logically associated data items in a NDM file. Records are taken as a whole. A program does not read, delete or update part of a record, but rather processes the entire record all at once.

Relation File

A relation file has one record for each relationship between files in an NDM application. There is typically one relation file per application. Refer to Chapter 3 for more information.

Return Code

Every NDM function sets a return code value that must be examined by the application. A value of zero indicates successful completion of the function. Refer to Chapter 11 for details.

Return Code File

The error code file contains a record for each return code generated by the NDM API. Any return code other than 0 is considered to be an error. Refer to Chapter 7 for more information.

Toolbox Feature

A native ISAM feature that is not supported by all NDM supported native ISAMs. A call to **31420 NDM_SET_TOOLBOX_STATUS** must be made to enable each feature that is to be used.

Unique Keys

Keys can be unique or non-unique. If a customer's zip code was used as a key field, this would be a non-unique key because many customers can have the same zip code. But, a customer's account number is usually a unique key, i.e. two customers should not have the same account number.

INDEX

!

\$CLOSE 8-4

\$OPEN 1-5, 8-4 - 8-5

A

Accessee File 3-16

Accessor File 3-16

Active Users 4-6

API Function Parameters

ALPHA_OR_NUMERIC\$ 10-37

API_REVS\$ 10-32

API_SERIAL\$ 10-32

B2C_TO_NDM\$ 10-58

BOF\$ 10-41, 10-66

CATALOG_HANDLES\$ 10-17, 10-30

CONVERSION_ENTRY_COUNT 10-32, 10-59

CONVERSION_TABLE\$ 10-3, 10-8 - 10-12, 10-28, 10-32, 10-34 - 10-35, 10-45 - 10-46, 10-59

CONVERSION_TABLE_COUNT 10-32, 10-34, 10-59

CONVERSION_TABLE_LIST\$ 10-3, 10-28, 10-34

CONVERSION_TABLE_NUMBER 10-12, 10-45 - 10-46

DD_HANDLES\$ 10-10, 10-12, 10-22, 10-43, 10-45, 10-55

DD_NAMES\$ 10-30

DIRECTION 10-8

EOF\$ 10-41, 10-66

ERROR_CODE 10-3, 10-29, 10-39 - 10-40

ERROR_DESCRIPTIONS 10-3, 10-29, 10-39

ERROR_TYPE 10-39

FEATURE_NUMBER 10-56, 10-75

FIELD_CLASS\$ 10-37

FIELD_COUNT 10-43, 10-45

FIELD_NAME\$ 10-12, 10-45 - 10-46

FIELD_OK\$ 10-43, 10-45

FIELD_TYPE 10-37

FILE_COUNT 10-32, 10-51, 10-59

FILE_HANDLES 10-5 - 10-7, 10-14, 10-17, 10-20, 10-26 - 10-27, 10-36, 10-41 - 10-42, 10-47, 10-53, 10-61, 10-63, 10-65 - 10-66, 10-68, 10-70, 10-72 - 10-73, 10-81

FILE_NAMES\$ 10-14, 10-24, 10-30, 10-41, 10-63,

FILE_TITLES 10-17, 10-30

FORMAT_SPEC\$ 10-3, 10-43 - 10-46

HANDLE_SIZE 10-3, 10-14, 10-16, 10-47, 10-52, 10-63 - 10-64

HIGHWATER_TABLE\$ 10-50

INDEX_NUMBER 10-3, 10-10, 10-14 - 10-15, 10-17 - 10-18, 10-20, 10-26, 10-36, 10-43 - 10-44, 10-48, 10-63 - 10-64, 10-72

ISAM_CODE 10-32

ISAM_NAMES\$ 10-32

ISAM_SPECIFIC\$ 10-14 - 10-15, 10-30, 10-41,

KEYDESC_HANDLES\$ 10-10, 10-22, 10-43, 10-48 - 10-49

KEYDESC_NAME\$ 10-30

KEYDESC_TABLE\$ 10-14 - 10-15, 10-20 - 10-22

KEY_IN_RECORD\$ 10-10, 10-43 - 10-44,

KEYNAME\$ 10-48

KEY_TABLE_SIZE 10-3, 10-22 - 10-23, 10-42, 10-49

LOCKS 10-4 - 10-6, 10-61, 10-65 - 10-66, 10-68 - 10-69, 10-81

MAX_ALPHA_LEN 10-43, 10-45

MODE\$ 10-14, 10-17, 10-41, 10-63

NAME\$ 10-10, 10-12, 10-14 - 10-15, 10-24, 10-30, 10-32, 10-41, 10-44 - 10-46, 10-48, 10-58, 10-63 - 10-64, 10-77

NDM_DIRECTORY\$ 10-32 - 10-33

NDM_TO_B2C\$ 10-58

NEW_STATUS 10-75

OPEN_FILE_COUNT 10-51

OPEN_FILE_LIST\$ 10-4, 10-51

POSITION 10-4, 10-27, 10-53 - 10-54, 10-66, 10-68 - 10-69, 10-71, 10-73 - 10-74, 10-81

POSITION_HANDLES\$ 10-53, 10-73

RECORD_BUFFER\$ 10-5 - 10-6, 10-8, 10-61, 10-65 - 10-66, 10-68, 10-70

RECORD_COUNT 10-41

RECORD_LENGTH 10-4, 10-14, 10-16 - 10-17, 10-19, 10-41, 10-55

RECORD_SIZE 10-32, 10-59

API Function Parameters Cont.

RETURN_CODE 10-5 - 10-81
 SEARCH_TYPES 10-65 - 10-66
 STATUS_TABLE 10-41 - 10-42
 TOOLBOX_STATUS 10-4, 10-50, 10-56 - 10-57,
 10-60, 10-75 - 10-76
 USER_COUNT 10-32
 USER_LIMIT 10-32

API Functions

31010 NDM_APPEND_UNIQUE_KEY_RE-
 CORD 5-12, 6-20, 8-4, 8-7, 8-9, 10-5, 10-61, 10-
 81
 31020 NDM_CLOSE_FILE 5-3, 6-4, 6-12 - 6-13, 6-
 15, 8-8, 9-16, 9-21, 10-7
 31030 NDM_CONVERT 3-7, 4-53, 5-8 - 5-9, 5-11,
 5-13, 6-17 - 6-19, 6-21, 8-8, 9-19 - 9-20, 10-5 -
 10-6, 10-8, 10-12, 10-61, 10-66, 10-70
 31040 NDM_CREATE_CONV_TA-
 BLE_FOR_KEY 3-12, 5-8, 5-10 - 5-14, 6-7, 6-
 15, 9-18, 10-8, 10-10, 10-28, 10-34, 11-18
 31050 NDM_CREATE_CONV_TABLE 3-5, 5-8 -
 5-11, 5-13, -15, 9-18, 10-8, 10-12, 10-28, 10-34,
 11-16 - 11-17
 31060 NDM_CREAT_FILE 10-14 - 10-16
 31070 NDM_CREATE_FILE_FROM_CATA-
 LOG 2-4 - 2-5, 2-8, 3-5, 3-12, 5-2, 5-6, 6-4, 6-11
 - 6-12, 8-8, 8-10, 10-17
 31080 NDM_CREATE_INDEX 2-4 - 2-5, 2-7, 2-9,
 5-2, 5-16, 10-20, 11-21
 31090 NDM_CREATE_KEY_TABLE 2-8, 3-5, 3-
 12, 6-13, 10-14 - 10-15, 10-22, 10-49
 31100 NDM_DELETE_FILE 4-11, 10-24
 31110 NDM_DELETE_INDEX 2-9, 5-2, 5-16, 10-
 26
 31120 NDM_DELETE_RECORD 2-4, 6-4, 6-22, 9-
 20, 10-27
 31130 NDM_DESTROY_CONVERSION_TA-
 BLE 5-13, 10-28
 31140 NDM_GET_ISAM_ERROR_CODE 2-11,
 10-29
 31150 NDM_GET_CATALOG_ENTRY 3-5, 5-5,
 6-7, 6-14, 9-18, 10-14 - 10-15, 10-30
 31160 NDM_GET_CONFIGURATION 2-12, 6-6,
 6-9 - 6-11, --9-17, 10-32
 31170 NDM_GET_CONVERSION_TA-
 BLE_LIST 10-34
 31180 NDM_GET_CURRENT_INDEX 2-5, 6-16,
 10-36, 10-72
 31200 NDM_GET_DEFAULT_FIELD_TYPE 8-
 10, 10-37

31210 NDM_GET_ERROR_DESCRIPTION 2-
 11, 9-21, 10-39
 31220 NDM_GET_FILE_STATUS 10-41
 31223 NDM_GET_FORMAT_SPEC_FOR_KEY
 3-6, 3-12, 10-43
 31225 NDM_GET_FORMAT_SPEC 10-45
 31230 NDM_GET_HANDLE_SIZE 6-6, 6-8, 10-
 14, 10-47, 10-63 - 10-64
 31240 NDM_GET_INDEX_NUMBER 10-48
 31250 NDM_GET_KEY_TABLE_SIZE 10-22, 10-
 42, 10-49
 31260 NDM_GET_LIMIT_HIGHTWATERS 5-15,
 10-50, B-3
 31270 NDM_GET_OPEN_FILE_LIST 10-51
 31280 NDM_GET_POSITION 5-16, 10-53, 10-73
 31290 NDM_GET_RE-
 CORD_LENGTH_FROM_DD 3-5, 6-13, 10-
 14, 10-55
 31310 NDM_GET_TOOLBOX_STATUS 10-56
 31315 NDM_GET_TRANSLATION_TABLE 10-
 58
 31320 NDM_INITIALIZE 5-3 - 5-4, 6-4, 6-6, 6-8, 6-
 10 - 6-11, 9-17, 9-20, 10-32 - 10-34, 10-47, 10-50
 - 10-51, 10-56, 10-59, 10-75, 11-9, 11-16 - 11-18
 31330 NDM_INSERT_RECORD 2-10, 5-2, 5-12, 6-
 4, 6-20 - 6-21, 8-8 - 8-10, 9-19, 10-6, 10-61, 10-81
 31340 NDM_OPEN_FILE 2-4 - 2-5, 2-10 - 2-11, 3-
 5, 5-2, 5-6, 5-14, 6-4, 6-11, 6-13 - 6-15, 8-5, 9-17
 - 9-18, 10-14, 10-24, 10-41, 10-47, 10-63
 31360 NDM_READ_RECORD_BY_KEY 6-17, 9-
 19, 10-65, 10-81, 11-7
 31370 NDM_READ_RECORD_BY_POSITION
 9-20, 10-68, 10-81
 31380 NDM_REWRITE_RECORD 2-4, 2-10, 5-3,
 5-12, 6-4, 6-21, 9-20, 10-6, 10-61, 10-70
 31400 NDM_SET_CURRENT_INDEX 2-4 - 2-5, 5-
 6, 6-16 - 6-17, 9-5, 9-17, 10-72
 31410 NDM_SET_POSITION 10-73
 31420 NDM_SET_TOOLBOX_STATUS 2-7, 5-6,
 5-14, 6-9, 6-11, 9-17, 10-75, 11-9, 11-16 - 11-17
 31425 NDM_SET_TRANSLATION_TABLE 4-6,
 4-46, 4-53, 10-77
 31430 NDM_TRANSACTION_ABORT 5-15, 5-
 17, 10-78, 10-80, 11-10, 11-13
 31440 NDM_TRANSACTION_COMPLETE 5-15,
 5-17, 10-60, 10-79 - 10-80, 11-10
 31450 NDM_TRANSACTION_START 5-15, 5-17,
 10-78 - 10-80, 11-10
 31460 NDM_UNLOCK_ALL_RECORDS 10-6,
 10-61, 10-66, 10-68, 10-81

Application Portability 5-1, 5-4 - 5-6

Application Program Interface 1-2

B

Basic-2C Field Types 1-4, 4-17, 5-6 - 5-8

Benefits 1-1

Btrieve 3-4, 5-12 - 5-13, 5-15, 11-12 - 11-13, A-14, B-3

C

C-ISAM 3-4, 3-13, 5-15, 10-32

Catalog File 3-1 - 3-5, 3-16, 4-7 - 4-8, 4-10, 5-5, 6-6 - 6-7, 6-9 - 6-12, 7-2, 8-9 - 8-10, 11-3 - 11-4, 11-20, A-2 - A-4

Configuration 1-6, 6-11

Conventions 3-2, 5-5, P-1

Conversion Table 5-8 - 5-10, 5-12 - 5-14, 7-2, 8-8 - 8-9, 9-18 - 9-20, 11-3 - 11-4, 11-6, 11-14 - 11-17, 11-20

Conversion Table Handles 5-13, 6-10, 10-34

Conversion Table Number 3-7, 5-9 - 5-10, 10-12, 10-45 - 10-46, 10-55, 11-14

Converting Applications 8-5

Current Index 2-4 - 2-5, 6-16, 11-2, 11-7

Current Record 2-4 - 2-5, 2-7, 6-18, 11-2, 11-7, 11-12

D

Data Buffer 5-8

Data Description File 3-5 - 3-6, 3-10, 4-17 - 4-20, 5-8 - 5-11, 6-13 - 6-14, 7-2 - 7-4, 11-3 - 11-4, 11-14, 11-19 - 11-20, A-2 - A-5, A-8 - A-9

Data Dictionary File 3-1, 4-38, 4-40 - 4-41, 6-1 - 6-2, 6-11 - 6-12, 6-14, 7-1 - 7-2, 8-2, 8-9, A-1 - A-3, A-19, A-22

Data Dictionary System File 4-6, 4-10, 7-1 - 7-2, 8-9 - 8-10,

Data File 1-5, 3-1, 4-7 - 4-8, 4-30 - 4-31, 4-35 - 4-38, 5-4, 6-2, 6-7, 6-10 - 6-11, 6-14, 7-1, 8-1 - 8-5, 8-8, 9-3, 9-12 - 9-13, 11-5 - 11-7, 11-9, 11-12, A-2 - A-3, A-5, A-12 - A-16, A-19, A-22, B-2 - B-3

Data Independence 1-5

Default Field Type 4-4, 4-15, 4-17

DEFFN' 2-2, 6-12, 6-14, 6-16 - 6-17, 6-19, 6-21 - 6-22, 8-6

Defining Indices 2-1, 2-8

Deleting Records 6-2, 6-22

Descending Segments 2-7

Duplicate keys 3-14 - 3-15

E

Error Code 2-10 - 2-11, 4-4, 4-12, 4-14 - 4-15, 10-20, 10-29, 10-39, 11-2, 11-6 - 11-7, 11-12

Error Description 4-12 - 4-15, 10-39

Error Description File 10-39

Extended Features 5-6 - 5-7, 6-6 - 6-7, 6-9, 11-16

External Call 1-2, 2-2, 9-2

F

Field Class 7-4

Field Name 3-3, 3-6, 3-12 - 3-13, 3-16, 4-22, 4-27, 4-46, 7-3, 7-5 - 7-6, 8-3, 10-12, 10-45, 10-48, 11-19, A-10

Field Type

Basic-2C 1-4, 4-17, 4-53, 5-6 - 5-8, 7-3, 8-3, 8-10
Native 4-17, 7-3 - 7-4, 8-10

Field Type Conversion 1-7, 5-1, 5-6, 5-12, 6-1 - 6-2, 6-7, 6-14, 6-16 - 6-17, 7-2, 8-3, 8-9, 9-3, 11-15

Field Type File 7-3, 11-21

File Handles 5-1, 5-3, 6-8 - 6-10, 10-51, 10-59

File Limits 5-15, 10-16, 10-18, 10-21, 10-50, 10-64, 11-16 - 11-17

File Location 3-2, 5-5
 File Locking 1-5, 5-2
 File Name 3-5, 3-16, 4-4, 4-7 - 4-8, 4-19, 4-25,
 4-31, 4-44 - 4-48, 5-14, 7-2, 9-8, 10-16, 10-
 19, 10-25, 10-64, 11-2, 11-7, 11-15, A-2 -
 A-5, A-8, A-14
 File Naming 3-2, 5-5
 File Title 3-2 - 3-3, 3-16, 4-4, 4-7 - 4-10, 4-19,
 4-25 - 4-26, 5-5, 6-2 - 6-3, 6-12, 6-14, 8-2,
 8-10, 10-17, A-5, A-8, A-14
 Filename Extensions 10-14 - 10-16, 10-24, 10-
 30, 10-63 - 10-64

G

GOSUB' 1-2, 2-2 - 2-3, 6-10 - 6-23, P-1

I

Indices

Concepts 2-1
 Defining 2-1, 2-8
 Name 3-16
 Number 2-5, 3-15, 5-11, 5-13, 6-14 - 6-17, 6-19 - 6-
 20, 9-12, 10-15, 10-18, 10-20, 10-26, 10-48, 10-
 63, 10-72, 11-16, 11-21
 Segments 1-5, 2-6 - 2-7, 2-9, 3-14, 4-27, 5-5 - 5-7, 5-
 10 - 5-11, 5-14 - 5-16, 6-2, 6-7, 8-4, 10-10, 10-15,
 10-18, 10-20, 10-43 - 10-44, 10-65, 11-2 - 11-4,
 11-9, 11-16 - 11-17, 11-21, A-10 - A-11, B-2 - B-
 3
 IQ 3-13 - 3-14, 4-2, 4-5, 4-38 - 4-41, 5-12, 6-20,
 8-4, 8-7, 8-9, 10-3, 10-5, 10-61 - 10-62, 10-
 81, 11-17 - 11-18, A-1 - A-3, A-19 - A-22
 ISAM Code 4-4, 4-7 - 4-8, 4-11, 4-44, 6-13, 11-
 5, B-3
 ISAM Concepts 2-1, 2-3

K

Key Compression 3-14

Key Description Table 2-6, 2-8 - 2-9, 3-13, 6-
 13, 10-20, 10-22, 10-41, 10-49, 11-21, B-2

Key Types

Ascending 2-7, 3-13, 5-16
 Descending 2-7, 3-13, 5-16, 10-66
 Non-unique 2-7 - 2-8, 3-14
 Unique 2-7 - 2-8, 3-14 - 3-15, 5-4, 5-13, 8-7, 9-10, 10-
 5 - 10-6, 10-53, 10-73, 11-7

L

Length Increment 7-4
 Limits Table 10-50, 11-16
 Lines Per Page 4-5, 4-45 - 4-46

M

Memory Requirements 6-8
 Modular Code 9-7, 9-14
 Multi-user Considerations 5-1

N

Native Field Types 3-9, 5-5 - 5-8, 5-12, 8-3, 8-
 10, 10-37
 Native ISAM 1-1 - 1-7, 2-3, 2-6 - 2-11,
 Native Operating System Files 1-4
 NDM Development Package 1-3, 1-7, 6-2, 9-1
 NDM Directory 4-2, 6-6, 6-9
 NDM Environment Variable 4-2, 4-6, 6-9, 7-
 2, 9-2, 10-33
 NDM RunTime Package 1-3 - 1-4

O

Open file 2-5, 5-3, 6-8, 6-10 - 6-11, 6-14, 9-4, 9-
 6, 9-18, 10-5, 10-7, 10-27, 10-36, 10-41, 10-
 47, 10-51, 10-53, 10-59, 10-61, 10-65,
 10-68, 10-70, 10-72, 11-11
 Opening a File 9-1, 9-3

P

Page Size 4-4, 4-7 - 4-9, 4-11, 6-13, 11-3, 11-13, B-3
 Parameter Sizes 6-6 - 6-7
 Path 2-4, 3-2, 4-10, 4-38 - 4-39, 5-5, 6-9, 11-7
 Performance Issues 1-7
 Portability 1-1, 1-7, 2-10, 5-1, 5-4 - 5-6, 5-14 - 5-15, 10-50, P-1
 Pre-allocation page 4-7 - 4-9
 Print Address 4-5
 Print Device 4-5, 4-45 - 4-46

R

Read Only Mode 10-14, 10-17, 10-63
 Reading from Start of File 9-16
 Reading Sequentially 11-6
 Record Length 4-5, 4-45 - 4-46, 5-15, 6-6, 6-8, 6-10, 6-13, 10-14, 10-55, 11-9, B-3
 Record Locking 1-4 - 1-5, 5-1 - 5-3, 8-5
 Relation File 3-2, 3-15 - 3-17, 4-5, 4-47, 6-1, 6-14, 7-2, 9-8, A-2 - A-3
 Return Code 1-6 - 1-7, 2-1, 2-3, 2-10 - 2-11, 4-14, 5-6, 6-10, 6-12 - 6-13, 6-16 - 6-17, 6-19 - 6-20, 7-5, 8-3, 9-1, 9-7, 9-14 - 9-21, 11-1 - 11-2, 11-5 - 11-6, 11-16, 11-21
 Return Code File 4-14, 7-5, 11-1, 11-5

S

SDtrieve 3-4, 5-12, 5-15, P-1
 Segment Number 3-14, 4-27, 10-10, 10-44, 10-48, A-10
 Segment Zero 3-15 - 3-16
 Selecting an Index 6-2
 Shutdown Routine 6-2, 6-11
 Startup Routine 6-2, 6-6 - 6-7, 6-9 - 6-10

T

Temporary Indices 2-8 - 2-9, 5-16, 10-26
 Toolbox Features 2-10, 5-14, 10-59 - 10-60
 Translation File 4-46, 4-50 - 4-51
 Translation Table 3-8, 4-46, 4-53, 7-6 - 7-7, 10-58, 10-77

U

Unique Index 2-7 - 2-8
 Updating Records 6-2, 6-21
 User Limit 4-6
 Utilities
 Access Catalog File 4-4, 4-7, 4-30, A-3, A-12
 Access Data Description File 4-5, 4-7, A-4
 Access Error Description File 2-12, 4-4
 Access Field Type File 4-4
 Access Key Description File 4-5, A-8
 Access Translation File 4-6
 Access User Data File 4-2, 4-5, 4-30 - 4-34, A-15
 Copy User Data File 4-2, 4-5, 4-35 - 4-37
 Display Program Information 4-2, 4-6, 4-49
 Install a New Catalog File 4-5
 Set NDMUTIL Options 4-2, 4-5, 4-45 - 4-46, 4-48